CES
School of Computational Engineering and Science
McMaster University

Virtlab 1.1

Virtual Material Testing Laboratory (version 1.1)

M. Eng. Report

by
Gonzalo Sánchez
sancheg@mcmaster.ca

Submitted to the
School of Computational Engineering and Science
McMaster University
Hamilton, Ontario, Canada
in partial fulfilment of the requirements for the degree
Master of Engineering
(project option)

Supervisor:
Dr. Spencer Smith
smiths@mcmaster.ca

Created: 2010-02-20 Gonzalo Sánchez
Updated: 2010-04-26 Gonzalo Sánchez
Doc. nº: 151

Abstract

This document reports the modifications and additions that were done to the Virtlab software package with the aim of leveraging its existing codebase and documentation to industrial grade, and the impact thereof. The improvement of the software qualities of *Maintainability*, *Performance, Portability, Reliability, Reusability*, *Understandability* and *Verifiability*, as well as the overall *Usability* improvement achieved, obtained by application of Software Engineering techniques such as *separation of concerns* and *information hiding*, are presented as a clear example of the benefits that Software Engineering principles (and supporting tools) can provide for Scientific Computing software development and maintenance.

TABLE OF CONTENTS

INDEX OF TABLES....9

INDEX OF FIGURES....11

[1] ACKNOWLEDGEMENTS....13

[2] REFERENCE MATERIAL....15

[2.1] Related Documentation....15

[2.2] Terminology....15

[2.3] Abbreviations and Acronyms....16

[2.4] Conventions....17

[2.5] Virtlab Project and Software Package Reference Information....18

[3] INTRODUCTION....19

[3.1] Virtlab Software Library and Virtlab Project....20

[3.2] Purpose of the Report....21

[3.3] Scope of the Report....22

[3.4] Related Documentation Overview....23

[3.5] Organization of the Document....24

[4] VIRTLAB PROJECT OVERVIEW....25

[4.1] Introduction....26

[4.2] Motivation....27

[4.3] Theoretical Background....28

[4.3.1] Continuum mechanics goal description....28

[4.3.2] Stress....29

[4.3.3] Strain....30

[4.3.4] Stress – strain relationship: constitutive equation....32

[4.3.5] Body – environment relationship: equilibrium equation....33

[4.3.6] Continuum mechanics problem statement....33

[4.3.7] Virtlab's computational approach to continuum mechanics problem....34

[5] VIRTLAB PROJECT EVOLUTION....37

[5.1] Synopsis....38

[5.1.1] Initial State (2007/09): Virtlab v. 0.1....38

[5.1.2] Final State (2010/03): Virtlab v. 1.1....39

[5.2] Statement....40

[5.2.1] Rationale....40

[5.2.2] Software Qualities....41

[5.2.3] Goals....42

[5.3] Report of Changes and Additions....43

[5.3.1] Building process....44

[5.3.1.1] Portability issues....44
[5.3.1.2] Automation issues....48
[5.3.1.2.1] MatCalc v. 0.1....48
[5.3.1.2.2] MatCalc v. 1.1....50
[5.3.1.2.3] MatGen v. 0.1....53
[5.3.1.2.4] MatGen v. 1.1....55
[5.3.1.3] Future work....58
[5.3.2] MatCalc's GUI....59
[5.3.2.1] MatCalc v. 0.1 GUI....59
[5.3.2.2] MatCalc v. 1.1 GUI....62
[5.3.2.3] Future work....67
[5.3.3] MatCalc's CLI....69
[5.3.4] MatGen....73
[5.3.4.1] Code generation statement test....73
[5.3.4.2] Future work....75
[5.3.5] Separation of Concerns and Information Hiding....76
[5.3.5.1] File Input and Output modules....76
[5.3.5.2] GUI and CLI modules....77
[5.3.5.3] stdout and stderr handling....77
[5.3.5.4] configset module....77
[5.3.5.5] Virtlab v. 1.1 restructure overview....79
[5.3.5.6] Future work....79
[5.3.6] Material Model Parameters Fitting Tool....81
[5.3.6.1] Tool overview....81
[5.3.6.2] Hooke & Jeeves Pattern Search method overview....82
[5.3.6.3] Design decisions and implementation overview....84
[5.3.6.4] Tool preliminary performance and results....85
[5.3.6.5] Future work....90
[5.3.7] Documentation....91
[5.3.7.1] Doxygen codebase documentation....91
[5.3.7.2] ChangeLog and TODO files....92
[5.3.7.3] INSTALL2 file....93
[5.3.7.4] Subversion commit log....93
[5.3.7.5] Other support text files....94
[5.3.7.6] Future work....94
[5.3.8] Virtlab tests and tools....95
[5.3.8.1] lists tool....95
[5.3.8.2] dset_diff tool....96
[5.3.8.3] configset_test utility....97
[5.3.8.4] FEM_test tool....98
[5.3.8.5] Future work....99

[6] SOFTWARE TOOLS....101

[6.1] Subversion....102
[6.2] Build Systems....104
[6.2.1] The GNU Build System....106
[6.2.1.1] Autoconf, Automake and Libtool....106
[6.2.1.2] The configure shell script....106
[6.2.1.3] Makefiles....108
[6.2.1.4] Goals....110
[6.2.1.5] User's perspective....111
[6.2.1.6] Developer's perspective....112
[6.3] Doxygen....115
[6.4] Valgrind....123

[6.5] System tracing tools 130

[7] CONCLUSIONS & FUTURE WORK/RESEARCH....135

[7.1] Conclusions....136

[7.2] Future work....139

[7.3] Future research....141

[8] REFERENCES....143

[9] APPENDIX....145

[9.1] MatCalc ChangeLog....146

[9.2] MatCalc TODO....155

[9.3] MatGen ChangeLog....161

[9.4] MatGen TODO....162

[9.5] INSTALL2....164

[9.6] matcalc_driver.txt....167

[9.7] add_experiment.txt....169

[9.8] add_material.txt....171

[9.9] generate_material.txt....177

[9.10] test_matcalc.sh....184

[9.11] test_EvalMapleStatement.cc....188

[9.12] MatCalc 1.1 file list (Doxygen)....191

[9.13] MatCalc 0.1 memory leak bug description and proposed fix....194

[9.13.1] Description....194

[9.13.2] Fix....195

[9.13.3] Additional information....195

[9.13.3.1] C++:....195

[9.13.3.2] C++ / Ruby:....195

[9.13.3.3] Ruby:....196

[9.14] Compilation, Assembly and Linking Overview....197

INDEX OF TABLES

Table 1: Related documentation 15
Table 2: Terminology.....15
Table 3: Abbreviations and Acronyms 16
Table 4: Conventions.....18
Table 5: Project Reference Information.....18

INDEX OF FIGURES

Fig. 1: Boundary valued problem with prescribed displacements u and an applied surface traction T ([CSMAK07], Figure 4, p. 24).....28

Fig. 2: Resultant force (Δf) on a small element (ΔS) of the surface (S) enclosing an arbitrary volume (V) about a point (P) within the material continuum occupying a region of space (Ω) ([SMC09], Figure 6, p.24).....29

Fig. 3: Stress tensor for a point within a body ([SMC09], Figure 6, p.24).....29

Fig. 4: The initial configuration of neighbouring points P0 and Q0 moving to their deformed configuration P and Q, respectively ([SMC09], Figure 8, p. 27).....30

Fig. 5: Yield function F, the hardening effect, and the plastic potential Q in stress space ([McC07], Figure 2.3, p. 22).…32

Fig. 6: Virtlab modules, dependencies, and inputs.....35

Fig. 7: MatCalc v. 0.1 GUI after RESET.....59

Fig. 8: MatCalc v. 0.1 GUI after RUN.....60

Fig. 9: MatCalc v. 0.1 stdout.....61

Fig. 10: MatCalc v. 1.1 GUI after RESET.....62

Fig. 11: MatCalc v. 1.1 GUI after RUN.....63

Fig. 12: MatCalc v. 1.1 independent resizing capabilities of left and right panels.....64

Fig. 13: MatCalc v. 1.1 graph section of the right panel.....65

Fig. 14: MatCalc v. 1.1 data table section of the right panel.....65

Fig. 15: MatcCalc v. 1.1 file input parsing progress shown in stdout.....66

Fig. 16: MatcCalc v. 1.1 information in stdout stream, formatted to fit in 80 columns.....66

Fig. 17: MatCalc v. 1.1 CLI usage message.....69

Fig. 18: MatCalc v. 1.1 input/output file sample.....70

Fig. 19: Sample Gnuplot script for plotting data from a MatCalc v. 1.1 data file.....71

Fig. 20: Gnuplot graph (data: line 17 in Fig. 19) showing the use of interactive measurements.....71

Fig. 21: MatCalc v. 1.1 CLI configuration printout.....72

Fig. 22: Virtlab files relationship overview, for the files affected by the separation of concerns applied to the codebase..80

Fig. 23: Sample search by Hooke and Jeeves pattern move algorithm. The table in Fig. 24 describes in detail the complete process.....83

Fig. 24: Tabular description of the sample search path in Fig. 23.....83

Fig. 25: Call graph for the Parameter Fitting Tool for MatCalc Material Models, generated by Doxygen.....84

Fig. 26: Viscoelastic material ‘experimental’ test data (blue jagged trace) produced with Octave by contamination of a closed form solution (red smooth trace) with random noise.....85

Fig. 27: The parameter fitting tool’s console output upon reading a MatCalc file containing the simulation setup and initial guesses.....86

Fig. 28: Hooke & Jeeves search progress sample.....87

Fig. 29: Search progress plot (X axis: ElasticE; Y axis: eta).....88

Fig. 30: Search progress plot (X axis: ElasticE; Y axis: eta) with custom algorithm....89
Fig. 31: FEM_test help message....98
Fig. 32: configure script sample....107
Fig. 33: Makefile example....109
Fig. 34: Key GNU build system tools and files, from a developer and a user perspective....114
Fig. 35: Doxygen graph legend....118
Fig. 36: input.cc call and caller graphs, generated by Doxygen....119
Fig. 37: Brief / detailed comments and #include dependency graph for matcalc.cc, generated by Doxygen....120
Fig. 38: matcalc.cc code listing, generated by Doxygen....121
Fig. 39: Virtlab v. 0.1 library memory leak, 'flooding' 97.9% of the computer's memory....128
Fig. 40: Compile, link and execute stages for running program (a process). 198
Fig. 41: The relocation record....199
Fig. 42: The object files linking process....200

[1] ACKNOWLEDGEMENTS

To John McCutchan for his excellent work – irrefutable proof of which is that an inexperienced programmer like me could actually do something useful with the sophisticated code he developed, and learn from it;

to Dr. Spencer Smith for his guidance, constant support, and encouragement − a keystone in my life as a student at Mac, and a turn point in my life in Canada;

to Dr. Tim Davidson and Laura Kobayashi M.Sc. for their assistance throughout my studies at CES Department − the compass and map that guided me in hitherto unknown territories;

to my parents for their lifelong unconditional moral and financial support − without whom I would not exist, in any imaginable sense;

and to my friends back in Uruguay, my friends and the Outdoors Club gang at Mac, my office mates at ITB115, and my house-mates and the crew at the Bean Bar in Westdale for providing a joyful environment for all of the above,

my most sincere gratitude; I am truly indebted.

gonzalo

April 2010
ITB115
CES Department
McMaster University
Hamilton, ON (Canada)

[2] REFERENCE MATERIAL

[2.1] Related Documentation

Document ID	*Description*
VL_REP	Virtlab 1.1 - Virtual Material Testing Laboratory version 1.1: MEng Report by Gonzalo Sánchez; May. 2010 (*this document*)
VL_TM	Virtlab Technical Manual
VL_UM	Virtlab User Manual
VL_SCD	Virtlab Source Code Documentation

Table 1: Related documentation

Please refer to sections 3.2, 3.3 and 3.4 for an overview of the purpose/contents of the documents

[2.2] Terminology

Expression	*Description*
software package	The complete software's file collection: codebase, documentation, build and test scripts, associated data files, etc.
software codebase	The software's codebase files (plus the build and test scripts, and associated data files); essentially, the software package contents less the documentation set.
software [product]	The application itself, as seen from an user's perspective.
accuracy	A measure of the absolute or relative error of an approximate solution computed by the software, compared to the 'true' solution.
maintainability	The effort required for a developer to locate and fix errors and to add features to an operational software package.
performance	A measure of how large a problem can be handled by a software product, and how quickly an answer can be found. Performance is related to how efficiently the internal resources of the computer are used.
portability	The effort required for a developer to transfer the software package from one operating environment to another.
reliability	The probability that a software product will meet its stated requirements under a given usage profile. As an example, an scientific computing application is reliable if the true error is rarely larger than the user requested bound on the error.
reusability	The extent to which a program/module of a software package can be used in other applications.
understandability	The degree to which a developer finds a software package easy to understand.
usability	The degree to which an end user of a software package finds the package easy to install and use.
verifiability	The effort necessary to verify the properties of a software package.

Table 2: Terminology

[SY09], s. 2 (except for: software package, software codebase, software [product])

[2.3] Abbreviations and Acronyms

Abbreviation / Acronym	*Reference*
Virtlab	**Virtual Material Laboratory**: A software package, which in essence constitutes an user extensible software library that implements a material modelling environment and material testing simulator. *The present report addresses the work done on the software package during the period May 2009 – March 2010.*
HTML	**Hyper Text Markup Language**: The current (and predominant) markup language for coding web pages, developed by the World Wide Web Consortium (W3C) & Web Hypertext Application Technology Work Group (WHATWG); requires a web browser for viewing.
XML	**Extensible Markup Language**: Encoding rules for electronic format documents. Defined in the XML 1.0 specification produced by the World Wide Web Consortium (W3C) and several other related specifications; all are fee-free open standards
PDE	**Partial Differential Equation**: An equation (or set of equations) involving an unknown function (or functions) of several independent variables, and its (or their) partial derivatives with respect to those variables.
FEM	**Finite Element Method** (also known as **Finite Element Analysis**): A technique used in the numerical computation of approximate solutions to PDE problems.
DSL	**Domain Specific Language**: A specification-oriented programming language, tailored to a particular problem domain, which allows one to express that domain's particular type of problems more easily and clearly than with a general-purpose programming language.
1D	**1 Dimensional**: Pertaining to a unidimensional object, either in the usual physical space, or in a mathematical space.
3D	**3 Dimensional**: Pertaining to a tridimensional object, either in the usual physical space, or in a mathematical space.
GUI	**Graphical User Interface**: Interface to an application running on a computing device, that enables the user to interact with the application by means of text entered by keyboards, and actions performed with haptic devices such as pointers; these usually require the system to have a screen that can display graphical information. *c.f.* CLI
CLI	**Command Line** (User) **Interface**: Interface to an application running on a computing device, that enables the user to interact with the application by text input from keyboard devices, and (primarily) text output to the computing system's output device (typically a screen) or a file. *c.f.* GUI
SC	**Scientific Computing**: the use of computers and programs with scientific and/or engineering purposes
SE	**Software Engineering**: the application of engineering principles (a systematic, disciplined, quantifiable approach) to the development, operation, and maintenance of software, and the study of these approaches.
SQ	**Software Qualities**: A collection of measures that together define the excellence or worth of a software product or process: *Accuracy*, *Maintainability*, *Performance*, *Portability*, *Reliability*, *Reusability*, *Understandability*, *Verifiability.*

Table 3: Abbreviations and Acronyms

Wikipedia [English] URL: http://en.wikipedia.org) (except for: Virtlab; SQ [SY09], s. 2)

[2.4] Conventions

Issue	*Convention*
References naming	Bibliographical references are indicated in square brackets, with the following criteria: References by a single author will be indicated with the first three letters of the author's surname, followed by the last two digits of the publication year of the referred work *e.g.* [Sur08]. Conflicting cases will be distinguished by appending a letter, *e.g.* [Sur08a], [Sur08b], etc. References by multiple authors will include each author's surname first letter, followed by the year; *e.g.* [AB08], [ABCD08]. Conflicting cases will be resolved similarly as for references by single authors. For references other than publications, *e.g.* software products/packages, the single author criteria will be applied, taking the original author of the software and the year of the first release of the software. Other cases will be dealt with by following as much as possible the aforementioned criteria.
References usage	Where appropriate, the cited reference will be accompanied by a chapter (ch.), section (s.), page (p.), and/or page range (pp.), indicating as precisely as possible the respective chapter, section, page, or page range being cited; in any of these cases, the complete reference citation will be presented within brackets, *e.g.* ([Sur08], s. 1.3), ([AB08], ch. 4, p.32), ([ABCD08], pp. 25-29).
Original/Previous References (References ancestry)	Cited references are those references actually read in the preparation of the present document. In case the cited references cite a previous/original reference or references which have not been read, the latter reference or references will *not* be cited herein. The rationale behind this scheme is that the citation will correspond to where the current author read/took the idea in its closest form to the actual use; it assumes that any referred authors will have in turn appropriately cited any previous reference to their own work.
Boldface / Italicized types	**Boldface** or *italics* are used in the text either for emphasis / highlighting purposes, or for indicating vector quantities. The different uses should be unambiguous by context. Within quotes, the original formatting of the text will be preserved as much as possible. Any deviation from the original, or any emphasis / highlighting that is *not* part of the original text, will be explicitly indicated in a footnote.
Fixed font types	`Fixed font` will be used for any code related text, such as code statements and code comments, as well as for reproducing program outputs to the system console (see 'Program outputs' below)
Sans-serif font types	Sans-serif font types (*i.e.* font types devoid of 'serifs' – small features at the end of the strokes – such as the Arial font) will be used to refer to elements in the GUI frontends to the software products, such as labels, or menu items. The purpose is to differentiate those text items from the rest of the text, which is rendered in a 'serif' font type (Times New Roman).
Emphasized paragraphs	Paragraphs that are deemed of particular importance, or requiring emphasis, will be centered, rendered on a grey background, and will have significantly reduced margins, as the following paragraph: Sample emphasized paragraph.

Program outputs	Program outputs, typically to the system console, which are not presented within a screenshot, will be rendered with fixed font and with a grey background (lighter than the emphasized paragraph's background) as the following paragraph:
	`Sample program output, typically to the system console,`
	`consisting of two lines of output.`

Issue	*Convention*
Verbatim quotes	Text that is quoted verbatim will be enclosed in quotation marks, and rendered with italicized font on a grey background similar to that of the emphasized paragraphs (see 'Emphasized paragraphs' above). Appropriate reference will be made after the closing quotation marks, and any particular observation concerning the quotation will be done on a footnote. *"Sample verbatim quotation, where the original text, having no particular emphasis, has been selectively emphasized by the current author with **boldface** type, fact that would be acknowledged on the associated footnote."*[1] ([Ref00], p. 1)

Table 4: Conventions

[2.5] Virtlab Project and Software Package Reference Information

Virtlab Project Stage	*Virtlab Software Version Evolution*	*Developer*	*Period*
1	0.0 to 0.1	John McCutchan	Up to September 2007
2	0.1 to 1.1	Gonzalo Sánchez	May 2009 – March 2010

Table 5: Project Reference Information

[1] Sample verbatim quotation paragraph footnote, acknowledging that the boldface type used for emphasis in the quoted text was *not* part of the original, and that the reference [Ref01] is a fake reference for example purposes.

[3] INTRODUCTION

This chapter of the Report provides an introduction to the Virtlab Software Library and Project, and states the purpose and scope of the Report, describes its relationship with the documentation listed in Table 1, and outlines the Report's organization.

[3.1] Virtlab Software Library and Virtlab Project

The Virtual Material Laboratory (Virtlab) is an extensible software library that implements a material modelling environment and material testing simulator. From a user defined material behaviour model and (virtual) material test, Virtlab will simulate the behaviour of the modelled material − that is, the material stress, strain, and deformation history are simulated − under the conditions imposed by the virtual material test settings [McC07]. The software library includes a default set of models for elastic, plastic, viscous, viscoelastic, viscoplastic, and elasto-viscoplastic materials[2], and a default set of load and displacement controlled virtual material tests[3]. Both material models and virtual tests are parametrized, and the corresponding parameters are set at run time. Additional material models and virtual tests can be added by the user to *extend* the software library as desired.

Virtlab software library was developed by John McCutchan for his M. Sc. Thesis, under the supervision of Dr. Spencer Smith at McMaster University [McC07]. It was originated as a 'proof of concept' software, being its underlying purpose that of applying software engineering techniques (such as commonality analysis, program families development, and model manipulation) to improve the qualities of a scientific computing software package: Accuracy, Maintainability, Performance, Portability, Reliability, Reusability, Understandability and Verifiability. At this stage, Virtlab Project's main objective was developing library modules that could be easily embedded in larger applications. This first stage was successfully finalized by September 2007 (see section 5.1.1).

Virtlab Project's second stage (May-August 2009; January-Marchl 2010; see section 5.1.2) shifted the objectives towards leveraging the software library qualities to 'industrial grade'. In this context, software engineering techniques (such as separation of concerns and information hiding) and tools (such as memory leak checkers, and automated code documentation generators) have been applied with the following results: the overall usability of the software package has been improved, by enhancing the building process and user interfaces; its portability has been extended from Linux and MacOS X to Solaris and OpenSolaris platforms; its understandability and maintainability has been significantly improved by adequate documentation; its performance has been improved by detection and correction of defects; its capabilities were extended by addition of tools, which in turn will aid to improve the accuracy and reliability of the software library; and its maintainability and reusability have been improved by addition and refactoring of library modules.

Virtlab Project, and its software product Virtlab Software Library, now constitute a non-trivial example case of Software Engineering principles and technologies application for the improvement of Scientific Computing software.

The Virtlab Project and software Library are described in greater detail in chapter 4, p. 25, and their evolution is reported in chapter 5, p. 37.

---

[2] 'Elastic' materials deform when loaded, and return to their original state when the load is removed. 'Plastic' materials yield after a loading threshold is reached, and will thereafter remain deformed even when the load is removed; the 'ductile' kind will withstand significant deformation, whereas 'brittle' ones will tend to fracture after yielding. 'Viscous' materials are those which will oppose resistance depending on the deformation rate to which they are subject to - typical behaviour in fluids. 'Viscoelastic', 'viscoplastic', and 'elasto-viscoplastic' materials present combinations of the aforementioned characteristic behaviours.

[3] In *load-controlled tests* a known load (force 'field') is applied over time to the test specimen, and the resulting specimen's deformation (strain state) is measured. In *displacement-controlled tests* the test specimen is subject to a known sequence of deformations over time, and the resulting specimen's internal loading condition (stress state) is measured.

[3.2] Purpose of the Report

This document constitutes the author's M. Eng. Report, to be submitted to the School of Computational Engineering and Science, McMaster University (Hamilton, Ontario, Canada) in partial fulfilment of the requirements for the degree of Master of Engineering (project option).

It is therefore intended for the examination committee, and in this context, its purposes are the following:

- *Describe* the Virtlab Project *origin*, *motivations,* and *goals.*
- *Report* the *modifications* and *additions* effected during the period May 2009 – March 2010 to the Virtlab software package, developed and maintained by McCutchan until September 2007 [McC07] (based on previous work of Smith and Gao, 2005, and Gao, 2004).
- *Describe* the *rationale* behind the modifications and additions to the software package during the period May 2009 – March 2010.
- *Evaluate* the *impact* of the modifications and additions to the software package during the period May 2009 – March 2010, and draw *conclusions*.
- *Suggest future work* for improving and enhancing the software package.
- *Suggest future research lines* that could use the software as a working example.

> The underlying purpose of the present Report is to provide an example that demonstrates the value of Software Engineering methods and technologies for Scientific Computing.

[3.3] Scope of the Report

In the context of the purposes described in section 3.2, the current report will address *what* was done, and *why* was that done, to the Virtlab software package during the period May 2009 – March 2010.

This report will *not* address *how* the changes and/or additions to the software package were accomplished, neither *how* the software internals work, nor *how* the software package should be built, installed, and used, except to the extent necessary to understand what was done, and why. The *how* issues are dealt with in other documents, a list of which is provided in Table 1 (section 3.4 provides an overview of the projected[4] contents).

The present report will assume the intended audience is neither familiar with the software package nor its application domain, and will henceforth present in chapter 4 an overview of the Virtlab project, which describes the software's functionality and capabilities, and provides the essential theoretical background for both the software and its application domain.

[4] To the date of writing of this document, **VL_REP**, of the remaining documents listed in Table 1 only the **VL_SCD** documentation set exists. For this reason, an extensive appendix has been added to this report, in which various of the currently existing documentation files − which will require appropriate inclusion or merging into the **VL_UM** and **VL_TM** documents − are listed as reference.

[3.4] Related Documentation Overview

The related documents listed in Table 1 provide the following information, *not* included in the present report:

- **[VL_UM] Virtlab User Manual**: The User's Manual is targeted to the end user of the software package; it provides instructions to *build* the software from the source code, describes how to perform the included *tests* in the software codebase, and describes how to *use* the software once compiled and installed on the user's computing platform.
- **[VL_TM] Virtlab Technical Manual**: The Technical Manual is targeted at developers and maintainers of the software package, as well as to those users who wish to evaluate its engineering/scientific foundations; it provides information on the *internal workings* of the software − such as its hierarchical and modular composition, modules interaction, and interactions with the software package dependencies[5] − as well as information of the software engineering tools/techniques applied in the development, test, and maintenance of the software package.
- **[VL_SCD] Virtlab Source Code Documentation**: The codebase documentation − generated by Doxygen [vHe97] from the software's source code files − is an organized collection of HTML (hyper-text markup language) files. The generated documentation uses syntax highlighting and/or colour coding for enhanced readability, and all objects therein are cross-referenced through hyper text links to facilitate browsing. The documentation set includes: complete source code files listing, index of data structures (class hierarchy and data fields) and folder/directory hierarchy, and indexes of all functions, structures, unions, enumerations, variables, type definitions, and definitions. Graphics are provided for all class inheritance diagrams, dependency graphs, and function call/caller graphs.

Note: To the date of writing of this document (**VL_REP**) of the remaining documents listed above (and in Table 1) only the **VL_SCD** documentation set exists. Therefore, an extensive appendix has been added to this report, in which the currently existing documentation files is listed for reference and preservation purposes. The information in this report's appendix will require appropriate inclusion into the **VL_UM** and **VL_TM** documents, when the latter are developed.

[5] From an end user's perspective, the software package currently requires the Ruby language run-time environment [Mat95] and − if automatic generation of the material model's code is desired − the Maple computer algebra system [Map80]. From a developers' standpoint, the software package additionally requires GNU Autotools suite [VETT00]. For the software package maintenance, Doxygen source code documentation generator [vHe97] (plus optional dependencies) is suggested, although not required. Details of these dependencies are provided in the VL_TM and VL_UM documents (please refer to Table 1).

[3.5] Organization of the Document

The first part of this document includes the **Table of Contents**, **Indexes of Tables and Figures, Acknowledgements**, the **Reference Material** tables (Related Documentation, Terminology, Abbreviations and Acronyms, Conventions) as well as the current **Introduction**.

The rest of the document, which constitutes the core of the Report, is organized as follows:

- **Virtlab Project Overview**: This chapter provides an overview of the Virtual Material Laboratory (Virtlab) Project and corresponding software package. It briefly describes the software's functionality and capabilities, states the underlying project's motivation and purpose, and provides the essential theoretical background for understanding the software package underpinnings and the report's contents.
- **Virtlab Project Evolution**: This chapter reports the project history, from its state as of September 2007, the evolution undergone in the period May 2009 to March 2010, and up to its current state. It is the core section of the Report, in which the goals for the modifications and additions introduced to the Virtlab software package are stated, the actual modifications and additions are reported, and the rationale behind these are described.
- **Software Tools**: This chapter presents an overview of key software tools that were used in the Virtlab Project. Most of the contents herein have been selected from the respective tools websites, manuals, or articles describing the tools usage. This reference information is intended for readers unfamiliar with the tools' purpose and/or basic usage. Examples of the use of these tools in Virtlab project are presented whenever appropriate.
- **Conclusions & Future Work/Research**: This chapter evaluates the effects of the modifications and additions effected in the software codebase during the period May 2009 – March 2010, and presents ideas for future work on the software package, as well as future research ideas based on the Virtlab project.
- **References**: This chapter lists the bibliographical references that are used throughout this document.
- **Appendix**: This chapter lists the contents of specific documentation files for the Virtlab software package, and is meant as a reference while the documents listed in Table 1 − where the information herein should be appropriately included − undergo development.

Note: The reader using the Portable Document Format (PDF) version of this document can take advantage of the hyper-text links provided for each of the cross-references within the document, such as: sections, footnotes, captions (figures and equations) numbering, corresponding page references, and bibliographical references.

[4] VIRTLAB PROJECT OVERVIEW

This chapter provides an overview of the Virtual Material Laboratory (Virtlab) Project and corresponding software package. It briefly describes the software's functionality and capabilities, states the underlying project's motivation and purpose, and provides the essential theoretical background for understanding the software package underpinnings and the report's contents.

[4.1] Introduction

Virtlab is an *extensible software library* that implements a *material modelling environment* and *material testing simulator*. From a user defined material behaviour model and (virtual) material test, Virtlab will simulate the behaviour of the modelled material − that is, the material stress, strain, and deformation history are simulated − under the conditions imposed by the virtual material test settings [McC07].

The software library includes a default set of models for elastic, plastic, viscous, viscoelastic, viscoplastic, and elasto-viscoplastic materials[6], and a default set of load and displacement controlled virtual material tests[7]. Both material models and virtual tests are parametrized, and the corresponding parameters are set at run time. Additional material models and virtual tests can be added by the user to *extend* the software library as desired.

The Virtlab software library consists of two programs: MatCalc, and MatGen[8]. MatGen is a material model generator, which takes a material model specification described in a simple DSL (Domain Specific Language) and uses the Maple computer algebra system [Map80] to perform symbolic computations and generate C code, which is then inlined in a C++ class that implements the material model. The set of these classes are linked at compile time with the material test simulator (MatCalc) to be available at run time. MatCalc's 3D FEM (Finite Element Method) simulation engine can then apply load or displacement controlled virtual material tests to the included material classes, to simulate the material's stress, strain, and deformation history under the specified virtual test conditions. The virtual material tests are classes described in the Ruby programming language [Mat95] in plain text files; MatCalc launches the Ruby interpreter to parse the files at run time, so virtual material tests can be added or modified anytime without the need to rebuild the software to make them available.

Both Virtlab main components, MatGen and MatCalc, provide GUI (Graphical User Interface) and CLI (Command Line Interface) frontend choices, that enable the users to access the underlying library's functionality in convenient ways, either for interactive use, or batch processing. The design of Virtlab as a software library also enables developers to embed the core library modules in other software packages.

[6] 'Elastic' materials deform when loaded, and return to their original state when the load is removed. 'Plastic' materials yield after a loading threshold is reached, and will thereafter remain deformed even when the load is removed; the 'ductile' kind will withstand significant deformation, whereas 'brittle' ones will tend to fracture after yielding. 'Viscous' materials develop stress depending on the deformation rate to which they are subject to - typical behaviour in fluids. 'Viscoelastic', 'viscoplastic', and 'elasto-viscoplastic' materials present combinations of the aforementioned characteristic behaviours.

[7] In *load-controlled tests* a known force is applied over time to the test specimen, and the resulting specimen's deformation is measured (from which the strain state can be calculated). In *displacement-controlled tests* the test specimen is subject to a known sequence of deformations over time, and the resulting specimen's internal loading condition (stress state) is calculated (see section 4.3 for an overview of continuum mechanics basic concepts).

[8] The Virtlab software library is compiled by default into two programs, MatCalc and MatGen, but is *not* limited to be used in such a way: its modules can be reused in the context of larger applications.

[4.2] Motivation

The key enabling factor for developing an extensible material behaviour simulator, such as Virtlab, results from a commonality analysis for a family of material models [SMC09]. The commonality analysis suggests that a *single* constitutive equation[9] can be used for a *variety* of materials: elastic, plastic, viscous, viscoelastic, viscoplastic, and elasto-viscoplastic materials. Hence, rather than developing individual programs to simulate the behaviour of each class of materials, the commonalities enable the development of a *program family* for material behaviour simulation, which improves the *usability* and *reusability* of the resulting program [SMC07].

However, the aforementioned common constitutive equation for the family of material models can lead to potentially complex constitutive equations for the family member instances, since the relationship between material stress and deformation can rely on multiple non-linear equations involving second order tensors. Therefore, developing the actual code that models each specific material is a difficult and error prone task that requires significant expertise in scientific computing and materials science.

Virtlab overcomes the problem of hand-coding the material model by using a *generative programming* approach, that actually takes advantage of the material's common constitutive equation. *Generative programming* is the practice of having one program write the source code for another program, where typically the input to the first program is a *simple* description of the code to be generated for the second program. Typically, the descriptions fed to the first program are conveyed in a DSL which − being domain specific by definition − makes it easier than using a general purpose programming language to 'manually' develop the code for the second program directly. In this context, the common constitutive equation provides an abstract material description from which the code for each specific member of the family can be generated. Hence, only the specific member properties need to be described for the code generation stage, and this task is accomplished with the DSL ([CSMAK07], pp. 29-30).

Therefore, underlying the motivation for the Virtlab project, there is a specific intention of applying software engineering techniques such as commonality analysis, program families development, and model manipulation (in this case, by means of symbolic transformations and code generation) to improve the *qualities* of this scientific computing software package: *Reliability*, *Accuracy*, *Performance*, *Understandability*, *Maintainability*, *Verifiability*, *Reusability*, *Portability*[10] ([SY09], s. 2).

[9] The *constitutive equation* for a material model relates the material's *stress* and *strain* tensor histories. See section 4.3 for a brief theoretical background on continuum mechanics, in which these terms are succinctly explained.

[10] These concepts are succinctly described in Table 2 (Terminology) and in section 5.2.2, p. 41 (Software Qualities).

[4.3] Theoretical Background

This section presents the essential theoretical background of Virtlab's software package computational approach to the continuum mechanics problem presented by material modelling and material testing simulation.

[4.3.1] Continuum mechanics goal description

*"**In continuum and computational mechanics the goal is often to solve for the changes of deformation and stress within a given body**, as depicted in Cartesian (x-y-z) space* [representation] *in* [Fig. 1]. *To solve for the histories of deformation and stress **it is necessary to satisfy the governing partial differential equation for equilibrium, subject to initial conditions and boundary conditions**. The initial conditions could involve setting initial stress or strain fields, while the boundary conditions could involve specifying the surface traction **T** or the prescribed displacements **u**. The equilibrium equation alone does not provide enough information (equations) to solve for all of the unknowns, so the constitutive equation is added. **The constitutive equation postulates a dependence of the stress on the history of deformation, which allows for a solution to the overall problem**."*[11] ([CSMAK07], p. 24)

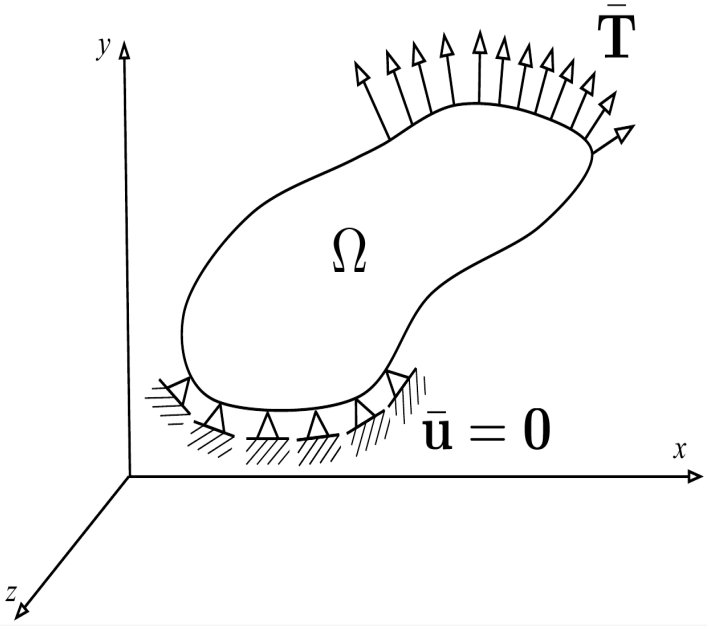


Fig. 1: Boundary valued problem with prescribed displacements **u** and an applied surface traction **T** ([CSMAK07], Figure 4, p. 24)

[11] The boldface type applied here if for 'highlighting' sections of particular relevance in the quote; it is *not* part of the original text.

[4.3.2] Stress

Stress is a measure of the *force per unit area* for each *direction* at a *point within the body* ([SMC09], p. 24):

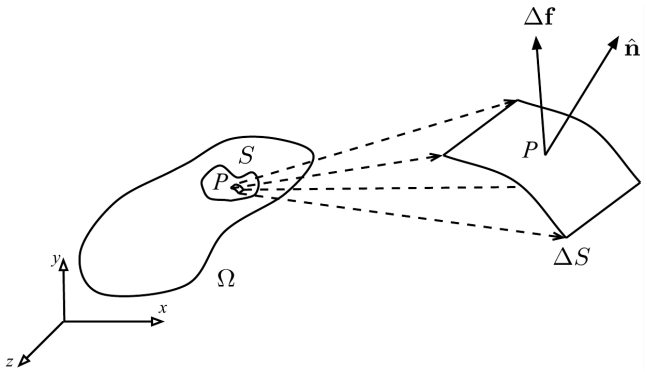


Fig. 2: Resultant force (Δ**f**) on a small element (Δ*S*) of the surface (*S*) enclosing an arbitrary volume (*V*) about a point (*P*) within the material continuum occupying a region of space (Ω) ([SMC09], Figure 6, p.24)

It is possible to summarize the *state of stress* **σ** *at a point within a body* by considering only three directions for **n**. For an arbitrarily chosen infinitesimal cube centred at the point within the body, this yields three traction forces acting on the faces of the infinitesimal cube ([SMC09], pp. 24-25):

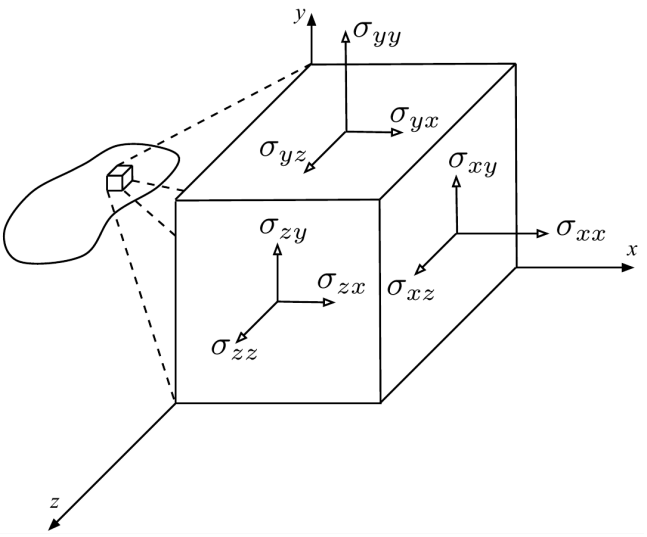


Fig. 3: Stress tensor for a point within a body ([SMC09], Figure 6, p.24)

The nine components of the state of stress **σ** depicted in Fig. 3 define the *stress tensor* which – under reasonable assumptions – is *symmetric*, implying that only six of its components are independent.

[4.3.3] Strain

Strain is a measure of the *deformation* undergone by a *body*, independent of the rigid rotations of the body. For the body shown at two different instants in time in Fig. 4, $|\,d\mathbf{x}\,|^2 - |\,d\mathbf{X}\,|^2$ is a convenient expression for the deformation that the body suffered during the considered interval ([SMC09], p. 25):

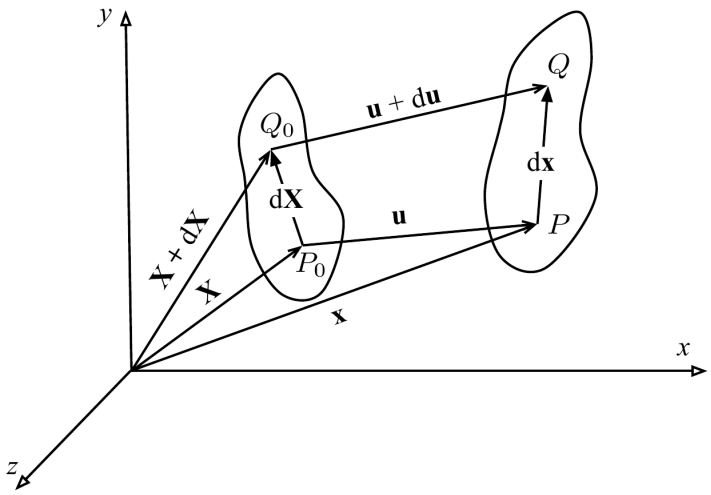


Fig. 4: The initial configuration of neighbouring points P_0 and Q_0 moving to their deformed configuration P and Q, respectively ([SMC09], Figure 8, p. 27)

The overall body deformation can be ultimately expressed as a *symmetric* nine element tensor ϵ which has only six independent components.

The relationship between the displacement **u** of a point and the corresponding strain state ϵ at the point is given by the *kinematic matrix* **L**:

$$\boldsymbol{\epsilon} = \mathbf{L}\cdot\mathbf{u} \;\; ; \;\; \mathbf{L}^{\mathrm{T}} \equiv \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial x} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}$$

Eq. 1: *Kinematic equation* relating the strain state ϵ at a point with its displacement **u**.

The *strain rate* of a body is then given by the corresponding symmetric nine element *strain rate tensor* $d\epsilon/dt$, which also has only six independent components ([SMC09], pp. 26-28). If the thermal effects are neglected (that is, the process is assumed to be isothermal) and *superposition* of elastic (ϵ^{e}) and viscoplastic (ϵ^{vp}) effects[12] can be assumed for modelling the material behaviour, then the strain rate $d\epsilon/dt$ undergone by a material can be expressed in terms of the *viscoplastic strain rate tensor* $d\epsilon^{vp}/dt$ as follows ([McC07], p. 21):

$$\dot{\boldsymbol{\epsilon}} = \dot{\boldsymbol{\epsilon}}^{e} + \dot{\boldsymbol{\epsilon}}^{vp}$$

Eq. 2: Additivity postulate of elastic and viscoplastic effects.

An expression for the viscoplastic strain rate tensor that allows modelling a wide variety of materials was proposed by Perzyna in 1966 ([McC07], p. 21):

$$\dot{\boldsymbol{\epsilon}}^{vp} = \lambda \cdot \frac{\partial Q}{\partial \boldsymbol{\sigma}} = \gamma \cdot \langle \varphi(F) \rangle \cdot \frac{\partial Q}{\partial \boldsymbol{\sigma}}$$

$$\langle \varphi(F(\boldsymbol{\sigma}, \kappa)) \rangle = \begin{cases} \varphi(F(\boldsymbol{\sigma}, \kappa)) & (F(\boldsymbol{\sigma}, \kappa(\boldsymbol{\epsilon}^{vp})) > 0) \\ 0 & (F(\boldsymbol{\sigma}, \kappa(\boldsymbol{\epsilon}^{vp})) < 0) \end{cases}$$

Eq. 3: Perzyna's viscoplastic strain rate tensor, in terms of a *fluidity parameter* γ, a *yield function* F, a *viscoplastic potential* Q, and a *hardening parameter* κ.

Perzyna's viscoplastic strain rate tensor is given in terms of a *fluidity parameter* γ, a *yield function* F (and an associated function φ) a *viscoplastic potential* Q, and a *hardening parameter* $\kappa(\epsilon^{vp})$ – dependent on the accumulated viscoplastic strain ϵ^{vp} – which determine the possible different material behaviours.

The viscoplastic strain rate tensor in Perzyna's model has *magnitude* given by $\lambda = \gamma \langle \varphi(F(\kappa,\boldsymbol{\sigma})) \rangle$, and a *direction* given by $\partial Q/\partial \sigma$ ([McC07], pp. 21-22):

- The *magnitude* depends on the surface defined by $F = 0$ in the six dimensional stress space, which is depicted as the innermost continuous line in the diagram in Fig. 5. Given the definition of F in Eq. 3, inside the surface the viscoplastic strain rate tensor $d\epsilon^{vp}/dt = 0$, and thus the material will actually behave in a purely elastic fashion, as per Eq. 2. If the stress history is such that the yield surface $F = 0$ is reached, the yield surface might change shape depending on the parameter $\kappa(\epsilon^{vp})$ which allows for the expansion or contraction of the yield surface. This allows modelling any *strain hardening* or *softening* behaviour of the material (as shown with a dashed line in the diagram in Fig. 5, for the hardening case).
- The *direction* is defined by the normal to the surface $Q = 0$ of the viscoplastic potential $Q(\boldsymbol{\sigma})$ (see Fig. 5).

[12] The elastic behaviour is typically modelled as a spring, whereas the viscoplastic behaviour is that of a material reluctant to sudden strain changes (viscous materials) and which deformation is permanent after the change in strain.

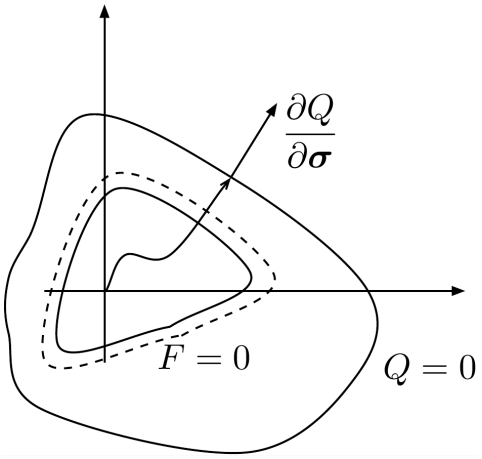


Fig. 5: Yield function F, the hardening effect, and the plastic potential Q in stress space ([McC07], Figure 2.3, p. 22).

[4.3.4] Stress – strain relationship: constitutive equation

The *relationship* between the the current *stress field* (Fig. 3) and the *deformation (strain) history* (Fig. 4) of the body is given by the *constitutive equation*, which is specific to the material.

Under the hypothesis of *isothermal* process in an *isotropic* material, a generalization of Hooke's law implies that the stress rate $d\boldsymbol{\sigma}/dt$ and the strain rate $d\epsilon/dt$ can be related by the *elastic constitutive matrix* $\mathbf{D}(E,\nu)$, in which the coefficients depend only on the material properties E (the elastic modulus[13]) and ν (Poisson's ratio)[14] ([McC07], p. 21):

$$\dot{\boldsymbol{\sigma}} = \mathbf{D}\,\dot{\boldsymbol{\epsilon}}^{e} = \mathbf{D}\left(\dot{\boldsymbol{\epsilon}}^{e} - \dot{\boldsymbol{\epsilon}}^{vp}\right)$$

Eq. 4: *Constitutive equation* (in *rate* form) − stress $\boldsymbol{\sigma}$ and strain ϵ related by the *constitutive matrix* $\mathbf{D}(E,\nu)$.

[13] Usually − and incorrectly − referred as Young's modulus. It should be attributed to Euler.

[14] The expression that defines the matrix **D** can be seen *e.g.* in [McC07], p. 20. For the purposes of this report, it suffices to know that the matrix is a constant coefficients matrix.

[4.3.5] Body – environment relationship: equilibrium equation

If at every instant the body is assumed to be in equilibrium[15], then – neglecting inertia, self-weight, and other body forces – the following *equilibrium equation*[16] must always be satisfied ([McC07], p. 18):

$$\underbrace{\begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial x} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix}}_{\mathbf{L}^{\mathrm{T}}} \cdot \underbrace{\begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{xy} \\ \sigma_{yz} \\ \sigma_{xz} \end{bmatrix}}_{\boldsymbol{\sigma}} = \mathbf{0}$$

Eq. 5: *Equilibrium equation*

[4.3.6] Continuum mechanics problem statement

For the problem and goal of continuum mechanics stated in section 4.3.1, and given:

- the *kinematic* equation Eq. 1
- a *constitutive* equation Eq. 4 that assumes *additivity of elastic and viscoplastic effects* as per Eq. 2 and models the *viscoplastic strain rate tensor* as per Eq. 3
- an *equilibrium* state as defined in Eq. 5

the continuum mechanics problem can be stated as follows:

To solve for the changes of deformation and stress within a given body, the following system of partial differential equations (PDE) need to be solved simultaneously, subject to *initial* and *boundary* conditions:

$$\left\{ \begin{aligned} \boldsymbol{\epsilon} &= \mathbf{L} \cdot \mathbf{u} \\ \mathbf{L}^{\mathrm{T}} \cdot \boldsymbol{\sigma} &= \mathbf{0} \\ \dot{\boldsymbol{\sigma}} &= \mathbf{D}(E, \nu) \cdot \left(\dot{\boldsymbol{\epsilon}} - \gamma \cdot \langle \varphi(F(\kappa(\boldsymbol{\epsilon}^{vp}), \boldsymbol{\sigma})) \rangle \cdot \frac{\partial Q}{\partial \boldsymbol{\sigma}} \right) \end{aligned} \right\}$$

Eq. 6: Continuum mechanics problem statement, in terms of the *kinematic equation,* the *equilibrium equation*, and the *constitutive equation* (based on Perzyna's model of the *viscoplastic rate tensor* and the additivity postulate of *elastic* and *viscoplastic* effects) which are described in the current section 4.3.

[15] This hypothesis is consistent with a typical real material mechanical test, in which the test specimen is kept in a 'static' position throughout the experiment.

[16] Where the stress tensor **σ** has been expressed as a 1D vector, where the first three components correspond to the *normal stresses* and the last three to the *shear stresses* of the infinitesimal cubic element.

[4.3.7] Virtlab's computational approach to continuum mechanics problem

Virtlab's approach to the continuum mechanics goal and computational problem statement described in section 4.3 is to have one of its components, MatGen, to deal with the coding aspects of the constitutive equation (Eq. 4) in terms of Perzyna's viscoplastic strain rate tensor model (Eq. 3), and leave to its other component, MatCalc, the actual numerical integration of the resulting system in Eq. 6.

At MatCalc's core, a FEM algorithm will solve the PDE system in Eq. 6:

> *"This method was selected because FEM naturally accommodates both displacement and load controlled boundary conditions. With the same finite element algorithm, all potential material tests can be simulated; all that changes between experiments is the input describing the boundary conditions. (...) Details of FEM can be found in Zienkiewicz et al. (2005). The derivation of the finite element equations for the viscoplastic constitutive equation follows the approach presented by Stolle (1991) (...) and mathematical manipulation (Smith, 2001)."* ([McC07], s. 2.3, pp. 23-24).

The input to MatGen is a description of the five *variabilities* of Perzyna's viscoplastic strain tensor model (Eq. 3) that is part of the (potentially complex) constitutive equation (Eq. 4)

1. the *fluidity parameter* γ,
2. the *yield function F*,
3. the *hardening parameter* $\kappa(\epsilon^{vp})$,
4. the *function* φ,
5. the *viscoplastic potential Q*,

and eventually some additional material specific constants required by the aforementioned parameters.

At MatGen's core, a client to the Maple computer algebra system calls Maple's symbolic computation engine to calculate the mathematical expressions in Eq. 6 (as well as other expressions required by the FEM algorithm in MatCalc) and calls Maple's code generation engine to actually generate the code for the computed expressions. MatGen then appropriately inlines the generated code in a C++ class, which defines the specific material model for the behaviour specified by the user in terms of γ, φ, F, κ, and Q.

The diagram in Fig. 6 shows the main components of Virtlab, its main dependencies, the main user inputs[17] and how these are related:

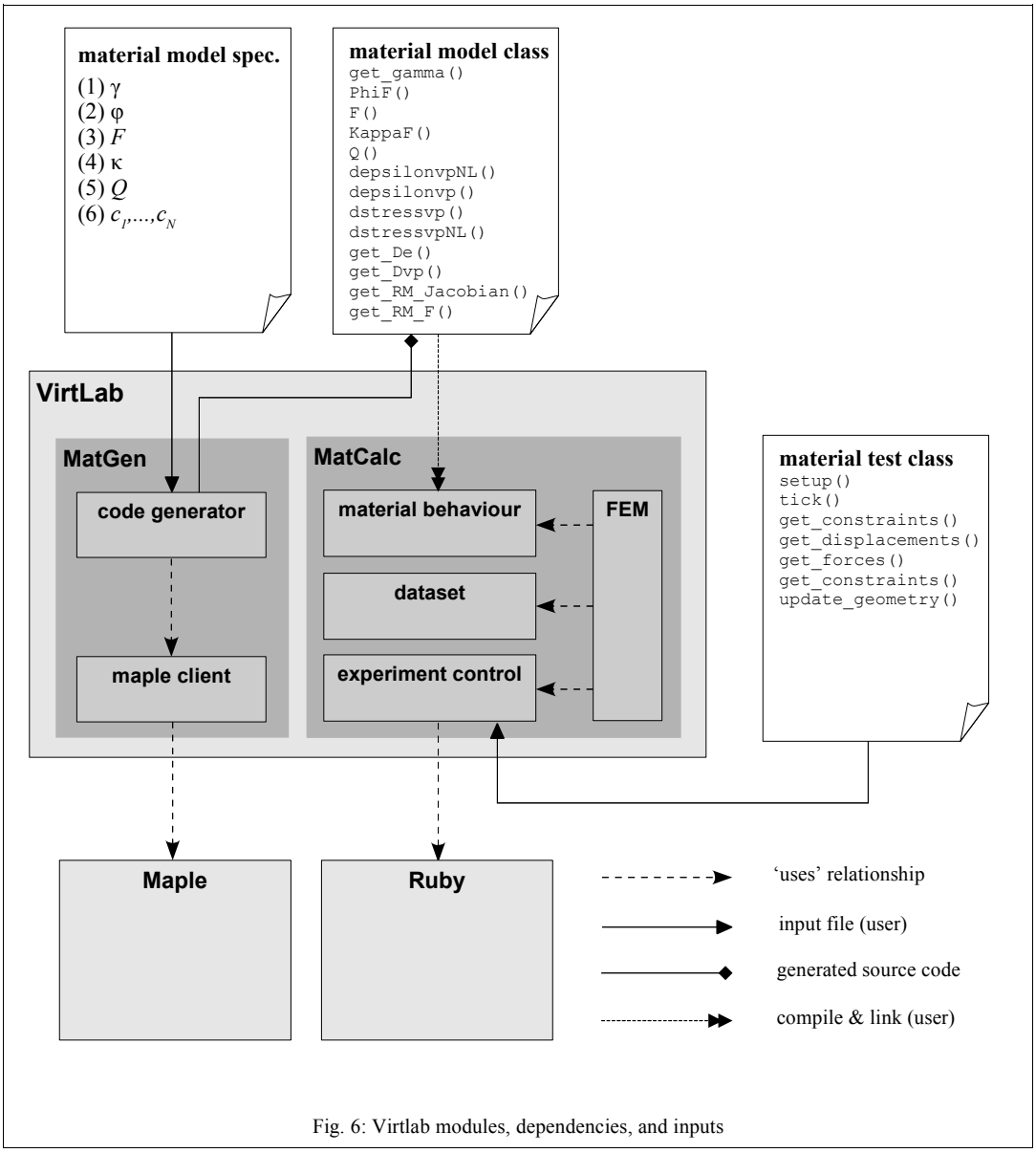


Fig. 6: Virtlab modules, dependencies, and inputs

[17] Except for the following user inputs, typically set via MatCalc's GUI frontend, or via a data input file for MatCalc's CLI frontend at run-time: time step, test duration, test specimen dimensions, material model, virtual test, and the values for the respective constants for the latter two.

[5] VIRTLAB PROJECT EVOLUTION

This chapter reports the project history, from its state as of September 2007, the evolution undergone in the period May 2009 to March 2010, and up to its current state. It is the core section of the Report, in which the goals for the modifications and additions introduced to the Virtlab software package are stated, the actual modifications and additions are reported, and the rationale behind these are described.

[5.1] Synopsis

This section provides a synopsis of the *initial* and *final states* of the *second stage* of Virtlab Project, which this report is addressing.

The initial state for the second stage of Virtlab Project is that of May 2009, which essentially is the state of the software package as of September 2007. The final state is defined at March 2010. However, it must be noted that there is ongoing work (scheduled until the end of April 2010) that is expected to improve the documentation of the FEM algorithm at the core of MatCalc, and the performance and documentation of the material model's parameters fitting tool. This is therefore classified under the 'future work' section (section 7.2, p. 139) of this report.

[5.1.1] Initial State (2007/09): Virtlab v. 0.1

Virtlab software library was developed by John McCutchan for his M. Sc. Thesis, under the supervision of Dr. Spencer Smith at McMaster University [McC07].

As stated in section 4.2, the Virtual Material Laboratory software library was foremostly intended as 'proof of concept' software, where the underlying purpose was that of applying software engineering techniques (such as commonality analysis, program families development, and model manipulation) to improve the qualities of a scientific computing software package: *Accuracy*, *Maintainability*, *Performance*, *Portability*, *Reliability*, *Reusability*, *Understandability*, *Verifiability*.[18]

At this stage, the project's main objective was developing library modules that a developer could easily embed in larger applications. So, despite the executables MatGen and MatCalc were created with the end user in mind, the frontends to these utilities were developed just to provide simple graphical user interfaces, which resulted neither altogether user friendly nor flexible enough to enable the potential end users to tap the full capabilities of the underlying library (as could be done by embedding its modules in other applications). Additionally, the scripts for building the original codebase into this two executables were not streamlined for a regular user: despite the fact that these configuration scripts were generated by the GNU 'Autotools' toolset[19] [VETT00] the main dependencies for MatGen and MatCalc (Maple and Ruby, respectively) had their locations 'hardcoded' into the build scripts, and even then, just addressed MacOS and Linux platforms, which were the developing platforms of choice. Finally, the only documentation for the codebase as of September 2007 were the embedded comments in the source code files, and the overall documentation for the software package in existence was that provided in the M. Sc. thesis corresponding to the original developer's work on the library [McC07].

> As of September 2007, Virtlab Software Library reached version 0.1.
>
> This is the *final state* of the software library at *completion* of the *first stage* of Virtlab Project, and thus the *initial state* for the *second stage* of Virtlab Project.

[18] These concepts are succinctly described in Table 2 (Terminology) and in section 5.2.2, p. 41 (Software Qualities).

[19] Refer to section 6.2.1, p. 106 for an overview of the GNU Build System.

[5.1.2] Final State (2010/03): Virtlab v. 1.1

At completion of Virtlab Project's second stage (March 2010) the graphical user interface for MatCalc has been significantly enhanced, providing a flexible and intuitive front-end to the library for setting up simulations, better graphical and text-based feedback controls for examining simulation runs, and file input/output capabilities to save/retrieve simulation parameters and data. These improvements − part of the *usability* quality improvements intended for the software package − enable the user to quickly experiment with material model parameters and virtual tests. A command-line interface has also been developed with the aim of performing 'batch' simulations (particularly in high-performance clusters) which not only adds to the *usability* purposes, but can potentially promote the *performance* quality, and pave the way to enhance the *portability* quality (*e.g.* enabling its compilation for use on systems devoid of graphical display capabilities but able to perform text-based and/or file input/output).

Code refactoring stages were effected with separation of concerns in mind, thus enabling both the GUI and CLI frontends to share the underlying library and auxiliary modules − the simulation engine is exactly the same for both frontends, and the input/output files from either interface are compatible with that of the other − thus addressing *reusability, understandability*, and *maintainability* qualities of the software package. The library itself benefits from this separation of concerns, primarily for the enhanced reusability of the modules.

The building process for the software package has been largely automated, and has also been extended to Solaris and OpenSolaris platforms, which addresses the *portability* quality in the first place, and as a consequence, its *reusability*, as well as indirectly contributing to its *reliability.*

The codebase documentation is currently in development, targeted to the Doxygen automated documentation generator [vHe97]; technical documentation of the whole project is also currently being developed, addressing the *understandability*, *reusability* and *maintainability* qualities.

A material model parameter fitting tool has also been developed, using the codebase as a software library. The tool uses the Hooke and Jeeves pattern search algorithm ([SHL83], s. 5.7) that runs the FEM engine iteratively to minimize a cost function that involves the sum of the squares of the differences between the simulation data and the real world data samples [San09]. This tool is currently being refactored to enable parallel computation [San10] by application of MPI constructs [Gro96].

> As of March 2010, Virtlab Software Library reached version 1.1.
>
> This is the *final state* of the software library at *completion* of the *second stage* of Virtlab Project.

[5.2] Statement

The work undertook as Virtlab Project's second stage on May 2009 was philosophically driven by the concept of *software qualities*[20] applied to a scientific software package. These qualities are founded in the rationale presented in the following section 5.2.1, and described in the next section 5.2.2. The project goals are derived thereafter, in section 5.2.3.

[5.2.1] Rationale

The rationale beneath the quest for software quality is clearly stated in [SY09], s. 1:

> *"The field of scientific computing (SC) has developed an impressive variety of algorithms and libraries that have greatly influenced and improved the work of scientists and engineers. SC software has, for instance, successfully been used to increase the productivity of manufacturing processes, to improve the effectiveness of health care treatments and to raise the level of safety obtained by new building and vehicle designs. However, instances have occurred where the confidence placed in SC software has been misplaced. For instance, Oliveira and Stewart [21] summarize three examples of SC software failures: (i) the 1991 failure of a Patriot missile to hit an incoming Scud missile; (ii) the 1996 explosion of the Ariane 5 rocket; and, (iii) the 1991 collapse of the Sleipner A oil rig. (...) The disturbing possibility of relying on incorrect SC software is further highlighted by the errors found when comparing nine large commercial packages for seismic data processing [14,15]. All of the packages were initially based on the same mathematical algorithms, but in experiments they produced different results, with an average absolute difference of 1% per 4000 lines of code [15]. The above examples illustrate that* ***problems exist with the software quality of correctness, even though correctness is generally considered to be the most important quality in SC*** *[19].* ***If this most important quality is not being achieved, then it is reasonable to believe that other software qualities, such as understandability, maintainability and reusability, are also suffering****."*[21,22] ([SY09], s. 1)

[20] These concepts are succinctly described in Table 2 (Terminology) and in section 5.2.2, p. 41 (Software Qualities).

[21] The boldface type applied here if for 'highlighting' sections of particular relevance in the quote; it is *not* part of the original text.

[22] **[14]** Hatton Les. *The chimera of software quality*. Computer 2007;40(8). **[15]** Hatton Les, Roberts Andy. *How accurate is scientific software?* IEEE Trans. Software Eng 1994;20(10):785-97. **[19]** Kelly Diane F, Sanders Rebecca. *Assessing the quality of scientific software*. In: Proceedings of the first international workshop on software engineering for computational science and engineering (SECSE 2008), Leipzig, Germany. In conjunction with the 30th international conference on software engineering (ICSE); 2008. **[21]** Oliveira Suely, Stewart David E. *Writing scientific software: a guide to good style*. New York (NY, USA): Cambridge University Press; 2006.

[5.2.2] Software Qualities

The *software qualities* referred to in this document are a *collection* of measures, that together define the excellence or worth of a software product or process ([SY09], s. 2):

- **[Q1] Reliability**: The probability that a software product will meet its stated requirements under a given usage profile. As an example, an scientific computing application is reliable if the true error is rarely larger than the user requested bound on the error.
- **[Q2] Accuracy**: A measure of the absolute or relative error of an approximate solution computed by the software, compared to the 'true' solution.
- **[Q3] Performance**: A measure of how large a problem can be handled by a software product, and how quickly an answer can be found. Performance is related to how efficiently the internal resources of the computer are used.
- **[Q4] Understandability**: The degree to which a programmer finds a software package easy to understand.
- **[Q5] Maintainability**: The effort required to locate and fix errors and to add features to an operational software package.
- **[Q6] Verifiability**: The effort necessary to verify the properties of a software package.
- **[Q7] Reusability**: The extent to which a program/module of a software package can be used in other applications.
- **[Q8] Portability**: The effort required to transfer the software package from one operating environment to another.

To the above qualities, an additional **Usability** quality can be added, that addresses the *overall easiness* with which an *end user* of the software package can install it, and make full use of it.

[5.2.3] Goals

With the mindset described in section 5.2.1, the Virtlab Project goals for the period May 2009 - March 2010 (Virtlab Project's second stage) were set as follows:

- Improve the *usability* of the software product: automate the build process as much as possible; improve the GUI frontend to MatCalc so that the full potential of the underlying library can be tapped; provide a CLI frontend to MatCalc to enable 'batch' simulations[23].
- Improve the *portability* and, consequently, the *reusability* qualities of the software product: extend it to Solaris/OpenSolaris platforms, which would render the software portable across almost all major Unix implementations used for scientific computing.
- Improve the software package *understandability* and *maintainability* qualities: generate appropriate documentation sets, oriented at users on the one hand, and to developers/maintainers on the other hand.
- Improve the software product's *accuracy, reliability* and *verifiability* qualities: develop a tool that allows the user to fit the material model parameters used for the simulation, to data obtained from real world material tests.
- Verify throughout the process if the software library qualities originally sought were effectively attained, and in that case, the benefits gained therefrom.

[23] MatGen was not considered in this usability quality improvements; its GUI can be improved but only slightly, and it already provides a command-line tool for 'batch' usage.

[5.3] Report of Changes and Additions

The following sections describe the main changes and additions to Virtlab software package version 0.1, done during Virtlab's Project second stage in the period May 2009 – March 2010, that leave the software package in version 1.1.

[5.3.1] Building process

From its inception, Virtlab library was built using the **GNU Build System** to best address the usability, portability and maintainability qualities of the software package.

> An overview of the GNU Build System is provided in section 6.2.1, p. 106, under the 'Build Systems' section 6.2.

The GNU build system has basically two goals ([GJ03], p. 1):

1. **Simplify the development of *portable* programs**.
2. **Simplify the building of programs that are distributed as *source code***.

The first goal is achieved by the automatic generation of a `configure` shell script[24], which enables an *end user* to compile and install the program directly from the source code distribution by a simple and automatic procedure.

The second goal is achieved by the automatic generation of Makefiles[25] and other shell scripts that are typically used in the building process, which facilitates the *developer's* tasks.

> Fig. 34 (p. 114) succinctly shows the key GNU build system tools and files, from a developer and a user perspective.

Despite the use of the GNU Build System in the Virtlab software package, the configuration instructions in the original versions of the `configure.ac` files[26] for MatCalc and MatGen were *not* streamlined for a regular user: the main dependencies − Maple and Ruby, respectively − had their locations 'hardcoded' therein, specifically for the original developer's platforms configuration. Additionally, the portability of Virtlab was limited to MacOS and Linux platforms, which were the development platforms of choice. Also, neither `make dist` nor `make install` Makefile targets worked as expected.

[5.3.1.1] Portability issues

Various portability issues have been successfully addressed by the use of the GNU Build System tools on Virtlab software library. These enabled to extend the platform options to Solaris and OpenSolaris.

For a start, the use of `autoscan` tool on Virtlab library suggested the addition of several checks in the `configure.ac` file. Among the suggested macros to include were:

- `AC_FUNC_STRTOD`
- `AC_TYPE_SIZE_T`
- `AC_C_INLINE`
- `AC_TYPE_SIGNAL`
- `AC_CHECK_FUNCS([pow])`

[24] See section 6.2.1.2, p. 106.

[25] See section 6.2.1.3, p. 108.

[26] See section 6.2.1.6, p. 112.

- `AC_CHECK_FUNCS([sqrt])`
- `AC_C_CONST`
- `AC_HEADER_STDBOOL`

Even though the software package could be built, installed, and used[27], solely by inspection of the names of the macro that the tool suggests adding, the developer is given an idea of the underlying portability issues at stake, which could lead to code refactoring to eliminate possible portability troubles for future platforms.

As an example, the following lines, extracted from the `configure` script generated by Autoconf from the `configure.ac` file corresponding to MatCalc, show the tests included by the `AC_CHECK_FUNCS([pow])` macro added therein:

```
# Checks for functions

for ac_func in pow
do
as_ac_var=`echo "ac_cv_func_$ac_func" | $as_tr_sh`
{ echo "$as_me:$LINENO: checking for $ac_func" >&5
echo $ECHO_N "checking for $ac_func... $ECHO_C" >&6; }
if { as_var=$as_ac_var; eval "test \"\${$as_var+set}\" = set"; }; then
  echo $ECHO_N "(cached) $ECHO_C" >&6
else
  cat >conftest.$ac_ext <<_ACEOF
/* confdefs.h.  */
_ACEOF
cat confdefs.h >>conftest.$ac_ext
cat >>conftest.$ac_ext <<_ACEOF
/* end confdefs.h.  */
/* Define $ac_func to an innocuous variant, in case <limits.h> declares $ac_func.
   For example, HP-UX 11i <limits.h> declares gettimeofday.  */
#define $ac_func innocuous_$ac_func

/* System header to define __stub macros and hopefully few prototypes,
    which can conflict with char $ac_func (); below.
    Prefer <limits.h> to <assert.h> if __STDC__ is defined, since
    <limits.h> exists even on freestanding compilers.  */

#ifdef __STDC__
# include <limits.h>
#else
# include <assert.h>
#endif

#undef $ac_func

/* Override any GCC internal prototype to avoid an error.
   Use char because int might match the return type of a GCC
   builtin and then its argument prototype would still apply.  */
#ifdef __cplusplus
extern "C"
#endif
char $ac_func ();
/* The GNU C library defines this for functions which it implements
    to always fail with ENOSYS.  Some functions are actually named
    something starting with __ and the normal name is an alias.  */
#if defined __stub_$ac_func || defined __stub___$ac_func
choke me
#endif

int
```

[27] Meaning the above checks were *in practise* unnecessary for the tested platforms.

```
main ()
{
return $ac_func ();
  ;
  return 0;
}
_ACEOF
rm -f conftest.$ac_objext conftest$ac_exeext
if { (ac_try="$ac_link"
case "(($ac_try" in
  *\"* | *\`* | *\\*) ac_try_echo=\$ac_try;;
  *) ac_try_echo=$ac_try;;
esac
eval "echo \"\$as_me:$LINENO: $ac_try_echo\"") >&5
  (eval "$ac_link") 2>conftest.er1
  ac_status=$?
  grep -v '^ *+' conftest.er1 >conftest.err
  rm -f conftest.er1
  cat conftest.err >&5
  echo "$as_me:$LINENO: \$? = $ac_status" >&5
  (exit $ac_status); } && {
       test -z "$ac_c_werror_flag" ||
       test ! -s conftest.err
       } && test -s conftest$ac_exeext &&
       $as_test_x conftest$ac_exeext; then
  eval "$as_ac_var=yes"
else
  echo "$as_me: failed program was:" >&5
sed 's/^/| /' conftest.$ac_ext >&5

      eval "$as_ac_var=no"
fi

rm -f core conftest.err conftest.$ac_objext conftest_ipa8_conftest.oo \
      conftest$ac_exeext conftest.$ac_ext
fi
ac_res=`eval echo '${'$as_ac_var'}'`
             { echo "$as_me:$LINENO: result: $ac_res" >&5
echo "${ECHO_T}$ac_res" >&6; }
if test `eval echo '${'$as_ac_var'}'` = yes; then
  cat >>confdefs.h <<_ACEOF
#define `echo "HAVE_$ac_func" | $as_tr_cpp` 1
_ACEOF

fi
done
   #(G.S. 2009/06/22)
```

As can be inferred from the above code snippet, the checks include information gathered out of years of experience of countless experts with different systems. It is *highly* unlikely that this complex and detailed information can be understood and handled by a single developer, without the aid of this kind of tool.

A more severe portability problem, that actually caused the `./configure` script to abort on Solaris and OpenSolaris platforms, was that of the `isinf` and `isnan` functions in Virtlab's `MC_test` module.

Since not all platforms use the same conventions, these functions had to be redefined by inserting in `MC_test.cc` the following code (suggested in the Autoconf tool manual (section 5.5.1, *Portability of C Functions*):

```
/* Autoconf: 5.5.1 Portability of C Functions [G.S. 2009/06/15] */

#ifndef isnan
# define isnan(x) \
    (sizeof (x) == sizeof (long double) ? isnan_ld (x) \
    : sizeof (x) == sizeof (double) ? isnan_d (x) \
    : isnan_f (x))
static inline int isnan_f  (float       x) { return x != x; }
static inline int isnan_d  (double      x) { return x != x; }
static inline int isnan_ld (long double x) { return x != x; }
#endif

#ifndef isinf
# define isinf(x) \
    (sizeof (x) == sizeof (long double) ? isinf_ld (x) \
    : sizeof (x) == sizeof (double) ? isinf_d (x) \
    : isinf_f (x))
static inline int isinf_f  (float       x) { return isnan (x - x); }
static inline int isinf_d  (double      x) { return isnan (x - x); }
static inline int isinf_ld (long double x) { return isnan (x - x); }
#endif
```

Although seemingly simple at first sight, a closer look at the above code suggests there might be quite subtle underpinnings to the C compiling standards being exploited therein to ensure portability among systems with compliant compilers. As said, it is highly unlikely that a single developer can understand and handle this kind of information without the aid from a specific tool.

After performing the aforementioned changes, MatCalc builds on Linux, MacOS, Solaris and OpenSolaris platforms.

Specifically, MatCalc was built and executed using the GNU Compiler Collection tool-chain on:

- Linux / x86_64
- MacOS X / x86_32
- Solaris 10 / UltraSPARC
- OpenSolaris 2009-06 / x86_32

[5.3.1.2] Automation issues

From the original codebase, neither of Virtlab's components, MatCalc and MatGen, could be built by just calling the corresponding `./configure` scripts without further command-line options, as is the goal of the GNU Build System. This stems from the fact that, despite the significant flexibility provided by the GNU build system, some dependencies do present a challenge for achieving complete building automation by generated `./configure` scripts.

Currently, only MatCalc's building process has been fully automated. MatGen's building process, although still requires user intervention, has been documented adequately to facilitate the task.

Additionally, neither `make dist` nor `make install` targets for MatCalc nor MatGen work as expected. Even though the packages are generated for distribution with `make dist`, these fail to build from the generated packages, so an end user cannot use the software. And even if built as a developer, by application of the whole GNU Build System toolchain, `make install` does *not* perform any action towards installing the generated executables in whichever location is specified at build time. This issue is *still unresolved.*

The following sections describe the original situation of the building process of MatCalc and MatGen, and the changes effected thereafter.

[5.3.1.2.1] MatCalc v. 0.1

The original Autoconf environment for MatCalc had a 'hardcoded' check for Ruby in the corresponding `configure.ac` file (which was specifically tailored to the original developer's platforms) since at the time of development automated checks for Ruby were unavailable:

```
# Check for ruby
AC_CHECK_HEADERS([/usr/lib/ruby/1.8/i486-linux/ruby.h],
[
        RUBY_CFLAGS="-I/usr/lib/ruby/1.8/i486-linux/"
        RUBY_LIBS="-lruby1.8"
],
[
        AC_CHECK_HEADERS([/usr/lib/ruby/1.8/universal-darwin8.0/ruby.h],
        [
                RUBY_CFLAGS="-I/usr/lib/ruby/1.8/universal-darwin8.0/"
                RUBY_LIBS="-lruby"
        ],
        [
                AC_CHECK_HEADERS([/usr/lib/ruby/1.8/powerpc-darwin8.0/ruby.h],
                [
                        RUBY_CFLAGS="-I/usr/lib/ruby/1.8/powerpc-darwin8.0/"
                        RUBY_LIBS="-lruby"
                ],
                [
                        AC_MSG_ERROR([libruby 1.8 is needed to compile])
                ])
        ])
])

AC_SUBST([RUBY_CFLAGS])
AC_SUBST([RUBY_LIBS])
```

For example, after generating the `configure` script with `aclocal`, `autoheader`, `automake`, and `autoconf`, the execution of the script yielded the following output on an OpenSolaris platform:

```
gonzalo@sancheg:~$ ./configure
checking build system type... i386-pc-solaris2.11
checking host system type... i386-pc-solaris2.11
checking target system type... i386-pc-solaris2.11
checking for a BSD-compatible install... /usr/bin/ginstall -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/gnu/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking dependency style of g++... gcc3
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for gcc option to accept ISO C89... (cached) none needed
checking dependency style of gcc... (cached) gcc3
checking for a BSD-compatible install... /usr/bin/ginstall -c
checking whether ln -s works... yes
checking whether make sets $(MAKE)... (cached) yes
checking for ranlib... ranlib
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /usr/gnu/bin/grep
checking for egrep... /usr/gnu/bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for char... yes
checking size of char... 1
checking for short... yes
checking size of short... 2
checking for int... yes
checking size of int... 4
checking for long... yes
checking size of long... 4
checking for void *... yes
checking size of void *... 4
checking for long long... yes
checking size of long long... 8
checking whether byte ordering is bigendian... no
```

```
checking for GTK... yes
checking for GLADE... yes
checking /usr/lib/ruby/1.8/i486-linux/ruby.h usability... no
checking /usr/lib/ruby/1.8/i486-linux/ruby.h presence... no
checking for /usr/lib/ruby/1.8/i486-linux/ruby.h... no
checking /usr/lib/ruby/1.8/universal-darwin8.0/ruby.h usability... no
checking /usr/lib/ruby/1.8/universal-darwin8.0/ruby.h presence... no
checking for /usr/lib/ruby/1.8/universal-darwin8.0/ruby.h... no
checking /usr/lib/ruby/1.8/powerpc-darwin8.0/ruby.h usability... no
checking /usr/lib/ruby/1.8/powerpc-darwin8.0/ruby.h presence... no
checking for /usr/lib/ruby/1.8/powerpc-darwin8.0/ruby.h... no
configure: error: libruby 1.8 is needed to compile
```

It can be seen in the last lines of the output produced by `configure`, that the script aborts upon being unable to find a working installation of Ruby in the system. The messages give evidence that the search is being done in *specific* locations; these correspond to the 'hardcoded' paths in the `configure.ac` file (the pertinent extract of which is shown above, before the `configure` script output).

Therefore, building the software package required either editing `configure.ac` file and rerunning the Autotools chain, or supplying the `RUBY_CFLAGS` and `RUBY_LIBS` variables on the `./configure` script command-line invocation, to override the aforementioned locations with those suitable for the particular platform in which the software was being built.

While passing arguments to the `./configure` script *might* be acceptable in some cases, it is in general tedious, so the use of macros that perform the specific dependency check is the preferred option. The GNU build system does provide facilities for cases where the developer might need to add a specific check to the build system.

[5.3.1.2.2] MatCalc v. 1.1

During the work on MatCalc during the period May 2009-March 2010, a macro for Autoconf that queries the system for an installed Ruby package and extracts the necessary information therein was found, and was therefore included in the `configure.ac` file:

```
# Check for ruby
AM_CHECK_RUBY(1.8)  #(G.S. 2009/07/03)
```

As with any locally supplied macro, the above macro call requires adding the line `ACLOCAL_AMFLAGS = -I m4t` to `Makefile.am` file, so that the GNU build system is informed of the existence of a locally supplied macro in the software package's `m4/` folder (that is, a macro that is to be provided with the software package, but is not part of the set of 'standard' macros distributed with Autoconf).

The aforementioned check for Ruby still required additional configuration lines to be added to `configure.ac` file, to deal with *other* portability issues:

```
# Link against libruby at compile time and runtime  #(G.S. 2009/07/03)
case "$host" in
    *-*-linux*)
        if test -e "$ruby_libdir/../../libruby.so" ; then
          AC_MSG_RESULT([*** found 'libruby.so' at: "$ruby_libdir/../../"])
          RUBY_LIBS="-L"$ruby_libdir/../../" \
```

```
                    -Xlinker -rpath -Xlinker "$ruby_libdir/../../" \
                    -lruby"
        else
          AC_MSG_ERROR([*** can't find 'libruby.so' at: "$ruby_libdir/../../"])
        fi
        ;;
    *-*-darwin*)
        if test -e "$ruby_libdir/../../libruby.dylib" ; then
          AC_MSG_RESULT([*** found 'libruby.dylib' at: "$ruby_libdir/../../"])
          RUBY_CFLAGS="-I\$(ruby_archdir)"
          RUBY_LIBS="-L"$ruby_libdir/../../" \
                    -lruby"
        else
          AC_MSG_ERROR([*** can't find 'libruby.dylib' at: "$ruby_libdir/../../"])
        fi
        ;;
    *-*-solaris*)
        if test -e "$ruby_libdir/../../libruby.so" ; then
          AC_MSG_RESULT([*** found 'libruby.so' at: "$ruby_libdir/../../"])
          RUBY_CFLAGS="-I\$(ruby_archdir)"
          RUBY_LIBS="-L"$ruby_libdir/../../" \
                    -R"$ruby_libdir/../../" \
                    -lruby"
        else
          AC_MSG_ERROR([*** can't find 'libruby.so' at: "$ruby_libdir/../../"])
        fi
        ;;
    *)
        AC_MSG_WARN([*** Check search and run paths for the linker if you have
libraries installed in non-standard locations. Refer to INSTALL2])
        ;;
esac
AC_SUBST(RUBY_CFLAGS)
AC_SUBST(RUBY_LIBS)
```

In particular, the above code fragment highlights the differences between Unix flavors when it comes to *linking* the software package to system or other dependencies libraries.

To the purpose of enhancing the usability, reusability, portability and maintainability qualities of the software package, an additional file `INSTALL2`, with installation hints that covers this specific compiling/linking issues, was generated and added to Virtlab software package. Section 9.5 in the Appendix lists the complete contents of the file, where compile time and run time linking issues are discussed.

After effecting all of the described changes, the building procedure for MatCalc was finally rendered fully automated:

```
gonzalo@sancheg:~$ ./configure
checking build system type... i386-pc-solaris2.11
checking host system type... i386-pc-solaris2.11
checking target system type... i386-pc-solaris2.11
checking for a BSD-compatible install... /usr/bin/ginstall -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/gnu/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
```

```
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking dependency style of g++... gcc3
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for gcc option to accept ISO C89... (cached) none needed
checking dependency style of gcc... (cached) gcc3
checking for a BSD-compatible install... /usr/bin/ginstall -c
checking whether ln -s works... yes
checking whether make sets $(MAKE)... (cached) yes
checking for ranlib... ranlib
checking whether gcc and cc understand -c and -o together... yes
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /usr/gnu/bin/grep
checking for egrep... /usr/gnu/bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for stdbool.h that conforms to C99... yes
checking for _Bool... yes
checking for dirent.h that defines DIR... yes
checking for library containing opendir... none required
checking for pow... no
checking for sqrt... no
checking for working strtod... yes
checking whether closedir returns void... no
checking for char... yes
checking size of char... 1
checking for short... yes
checking size of short... 2
checking for int... yes
checking size of int... 4
checking for long... yes
checking size of long... 4
checking for void *... yes
checking size of void *... 4
checking for long long... yes
checking size of long long... 8
checking whether byte ordering is bigendian... no
checking for inline... inline
checking for size_t... yes
checking return type of signal handlers... void
checking for an ANSI C-conforming const... yes
checking for GTK... yes
checking for GLADE... yes
checking for CPPUNIT... yes
checking for ruby... /usr/bin/ruby
checking for Ruby - version >= 1.8... yes
*** found 'libruby.so' at: "/usr/ruby/1.8/lib/ruby/1.8/../../"
configure: creating ./config.status
```

```
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating matcalc-config.h
config.status: executing depfiles commands
```

Note how the `configure` script is checking the following dependencies of MatCalc:

```
checking for GTK... yes
checking for GLADE... yes
checking for CPPUNIT... yes
checking for ruby... /usr/bin/ruby
checking for Ruby - version >= 1.8... yes
```

In case of failing to find an installed dependency, the build will abort with a corresponding error message, as it was shown for the case when Ruby could not be found.

If the dependency that `configure` is unable to locate is installed, then – similarly to the aforementioned commented problem with Ruby – either a suitable check for the dependency has to be added to `configure.ac` (and the complete Autotools chain rerun, as a developer would do) or an appropriate flag should be given in the `configure` script command-line invocation (as shown for MatGen v. 1.1 in section 5.3.1.2.4, p. 55). This requires the user to be knowledgeable of the software installed in his/her platform.

If the dependency that `configure` is unable to locate is *not* installed, then the user has to install it first. The dependency itself might be distributed as source code, and in that case, perhaps using the GNU Build System for compilation. Then, the user would require running the dependencies' `configure` script to build the dependency first, and then rerun MatCalc's `configure` script.

[5.3.1.2.3] MatGen v. 0.1

The original Autoconf environment for MatGen had a 'hardcoded' check for Maple in the corresponding `configure.ac` file (which was specifically tailored to the original developer's platforms) but the dependency is, in this case, 'hardcoded' *depending on the platform*, so the latter issue is the one that actually breaks the build, due to the *platform* not being found.

MatGen's `configure.ac` file shows the 'hardcoding':

```
case "$host" in
        *-*-mingw*|*-*-cygwin*)
                AC_DEFINE(PLATFORM_WIN32, 1, [Platform is Win32])
                AC_MSG_ERROR([Win32 not supported])
                ;;
        *-*-linux*)
                AC_DEFINE(PLATFORM_LINUX, 1, [Platform is Linux])
                MAPLE_ROOT="/opt/maple9.5"
                OPENMAPLE_LDFLAGS="-L\"$MAPLE_ROOT/bin.IBM_INTEL_LINUX\""
                ;;
        *-*-darwin*)
                AC_DEFINE(PLATFORM_APPLE, 1, [Platform is Apple])
                MAPLE_ROOT="/Applications/Maple 9.5/Maple 9.5.app/Contents/MacOS"
                OPENMAPLE_LDFLAGS="-L\"$MAPLE_ROOT/bin.APPLE_PPC_OSX\""
                ;;
        *)
                AC_MSG_WARN([*** Please add $host to configure.ac checks!])
```

```
                ;;
esac

OPENMAPLE_CFLAGS="-I\"$MAPLE_ROOT/extern/include\""
OPENMAPLE_LIBS="-lmaplec -lmaple -lgmp"
AC_SUBST([OPENMAPLE_CFLAGS])
AC_SUBST([OPENMAPLE_LDFLAGS])
AC_SUBST([OPENMAPLE_LIBS])
```

The corresponding `configure` script output on OpenSolaris is:

```
gonzalo@sancheg:~$ ./configure
checking build system type... i386-pc-solaris2.11
checking host system type... i386-pc-solaris2.11
checking target system type... i386-pc-solaris2.11
checking for a BSD-compatible install... /usr/bin/ginstall -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/gnu/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking dependency style of g++... gcc3
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for gcc option to accept ISO C89... (cached) none needed
checking dependency style of gcc... (cached) gcc3
checking for a BSD-compatible install... /usr/bin/ginstall -c
checking whether ln -s works... yes
checking whether make sets $(MAKE)... (cached) yes
checking for ranlib... ranlib
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /usr/gnu/bin/grep
checking for egrep... /usr/gnu/bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for char... yes
checking size of char... 1
checking for short... yes
checking size of short... 2
checking for int... yes
checking size of int... 4
```

```
checking for long... yes
checking size of long... 4
checking for void *... yes
checking size of void *... 4
checking for long long... yes
checking size of long long... 8
checking whether byte ordering is bigendian... no
checking for GTK... yes
checking for GLADE... yes
configure: WARNING: *** Please add i386-pc-solaris2.11 to configure.ac checks!
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: executing depfiles commands
```

[5.3.1.2.4] MatGen v. 1.1

Contrary to MatCalc's building process, at the time of writing this report, MatGen's building process has *not* yet been fully automated.

MatGen's main dependency location, the Maple computer algebra system, still needs to be specified by the user by passing to MatGen's `./configure` script certain environment variables.

A reminder has been included in the corresponding `configure.ac` file which alerts the user of this issue, upon running the `./configure` script. The following code snippet shows the warnings for the case of Solaris/OpenSolaris platforms:

```
# Warning for Maple dependencies  #(G.S. 2009/07/08)
case "$host" in
    *-*-solaris*)
        AC_DEFINE(PLATFORM_SOLARIS, 1, [Platform is Solaris/OpenSolaris])
        AC_MSG_WARN([*** Please specify Maple environment on the command line:])
        AC_MSG_WARN([*** MAPLE_ROOT="/path/to/maple/folders"])
        AC_MSG_WARN([*** OPENMAPLE_CFLAGS="-I\$MAPLE_ROOT/extern/include"])
        AC_MSG_WARN([*** OPENMAPLE_LDFLAGS="-L\$MAPLE_ROOT/bin.*SOLARIS*"])
        AC_MSG_WARN([*** OPENMAPLE_LIBS="-lmaplec -lmaple -lgmp"])
        ;;
    *)
        AC_MSG_WARN([*** Please add $host to configure.ac checks!])
        ;;
esac

AC_SUBST([OPENMAPLE_CFLAGS])
AC_SUBST([OPENMAPLE_LDFLAGS])
AC_SUBST([OPENMAPLE_LIBS])
```

The following command gives an example of the use of the appropriate linker switches, for configuring MatGen on a specific Linux platform (the Computing and Software Department at McMaster University server, `mills.cas.mcmaster.ca`[28]):

```
$ ./configure                                                          \
   OPENMAPLE_CFLAGS="-I/usr/local/maple11/extern/include"              \
   OPENMAPLE_LDFLAGS="-L/usr/local/maple11/bin.X86_64_LINUX            \
       -Xlinker -rpath -Xlinker /usr/local/maple11/bin.X86_64_LINUX" \
   OPENMAPLE_LIBS="-lmaplec -lmaple -lgmp"
```

[28] At the time of writing the present report. Any changes in the version or location of Maple in the referred server should be updated appropriately in the given command.

The corresponding `configure` script output on the said Linux server is:

```
sancheg@mills:~$ ./configure \
OPENMAPLE_CFLAGS="-I/usr/local/maple11/extern/include" \
OPENMAPLE_LDFLAGS="-L/usr/local/maple11/bin.X86_64_LINUX -Xlinker -rpath -Xlinker
/usr/local/maple11/bin.X86_64_LINUX" \
OPENMAPLE_LIBS="-lmaplec -lmaple -lgmp"
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking dependency style of g++... gcc3
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for gcc option to accept ANSI C... (cached) none needed
checking dependency style of gcc... (cached) gcc3
checking for a BSD-compatible install... /usr/bin/install -c
checking whether ln -s works... yes
checking whether make sets $(MAKE)... (cached) yes
checking for ranlib... ranlib
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking how to run the C preprocessor... gcc -E
checking for egrep... grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for stdbool.h that conforms to C99... yes
checking for _Bool... yes
checking for dirent.h that defines DIR... yes
checking for library containing opendir... none required
checking for strdup... yes
checking whether closedir returns void... no
checking for char... yes
checking size of char... 1
checking for short... yes
checking size of short... 2
checking for int... yes
checking size of int... 4
checking for long... yes
checking size of long... 8
```

```
checking for void *... yes
checking size of void *... 8
checking for long long... yes
checking size of long long... 8
checking whether byte ordering is bigendian... no
checking for size_t... yes
checking return type of signal handlers... void
checking for an ANSI C-conforming const... yes
checking for GTK... yes
checking for GLADE... yes
configure: WARNING: *** Please specify Maple environment on the command line:
configure: WARNING: *** MAPLE_ROOT="/path/to/maple/folders"
configure: WARNING: *** OPENMAPLE_CFLAGS="-I$MAPLE_ROOT/extern/include"
configure: WARNING: *** OPENMAPLE_LDFLAGS="-L$MAPLE_ROOT/bin.*LINUX*"
configure: WARNING: *** OPENMAPLE_LIBS="-lmaplec -lmaple -lgmp"
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: executing depfiles commands
```

Note that despite the pertinent flags have been specified in the `configure` script command-line, the warnings are still issued.

It is worth to remark again that the different compilers/assemblers/linkers require different flags and passing conventions to achieve the same goals – in this case, so that the linker can find the dependencies at compile time, *and* at run time. Please refer to section 9.5 in the Appendix, where the compile time and run time linking issues are discussed.

As a final remark on the building process changes: it should be clear from all of the above that **portability is *not* an easy quality to achieve – not even with a specific build system that was devised to address that need**. Reusability is therefore another software quality that might be affected by cross-platform issues.

[5.3.1.3] Future work

Fixes and enhancements to the building process of both MatCalc and MatGen should include:

- Diagnose and fix the problems that prevent `make dist` to generate proper packages of MatCalc and MatGen for the end users.
- Diagnose and fix the problems that prevent `make install` to properly install MatCalc and MatGen packages at the end user's specified location (using `–prefix=/path/` as option to `./configure`).
- Provide support for running checks at the building stage with `make check` (see section 5.3.8.5, p. 99).
- Embed in the building process the location of the XML definition files for both GUI frontends, to eliminate the 'hardcoded' relative path currently in effect[29] (see section 5.3.2, p. 59).
- (MatGen) Automate Maple dependency checks.
- Provide a `--without-gui` option in both `./configure` scripts, that would enable the user to build the software package in systems devoid of Gtk+ dependencies, or interested solely on the CLI frontends to the library (see section 5.3.2, p. 59).
- (MatCalc) Provide an option to build the material model class files as shared objects, loadable at runtime.

[29] Tests for installation in different system locations should be performed.

[5.3.2] MatCalc's GUI

The GUI frontend for MatCalc is built on the Gtk+ toolkit[30] [KP97]. Gtk+ is implemented in C, but seeking portability to all Unix flavors it does not rely on the standard `libc` library (that can vary between implementations) and therefore provides its own `Glib` library which replaces a number of standard calls. Gtk+ has C++ bindings that enable calling the Gtk+ functions from a C++ program, but its C interface might be used from within C++ programs by restricting the program to the C subset of C++, or by adopting a few conventions[31].

The user interface layout is actually defined in one Extensible Markup Language (XML) file, generated by the Glade Interface Designer (a GUI builder for Gtk+) [Gla98]. This allows to modify the layout of the interface independently of the underlying GUI code (as long as the cross-referenced widgets remain the same).

[5.3.2.1] MatCalc v. 0.1 GUI

The original GUI, for MatCalc v. 0.1, is shown in Fig. 7.

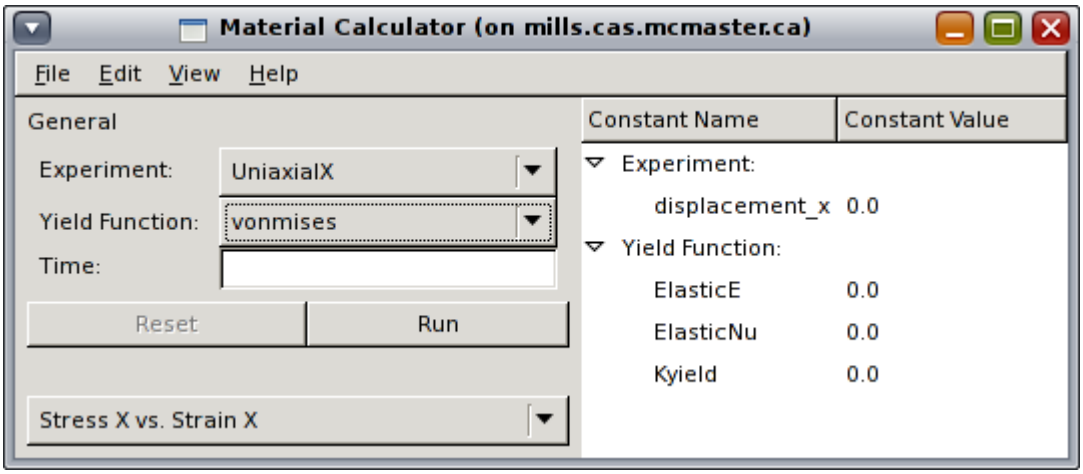


Fig. 7: MatCalc v. 0.1 GUI after RESET

The GUI's left panel allows the user to select the virtual material test ('Experiment' combobox) to apply to a selected material model ('Yield Function' combobox) during a specified time ('Time' edit box). On the right panel, the particular constants for the selected material test and model are unfolded[32], and the user is allowed to edit the values prior to running the simulation ('Run' button on the left side panel).

[30] The same Gtk+ toolkit is used for MatGen.

[31] Please refer to: http://library.gnome.org/devel/gtk-tutorial/stable/ and http://library.gnome.org/devel/gtk/stable/

[32] These constants vary according to the selected material test and model.

Fig. 8 shows MatCal's v. 0.1 GUI frontend after a simulation run:

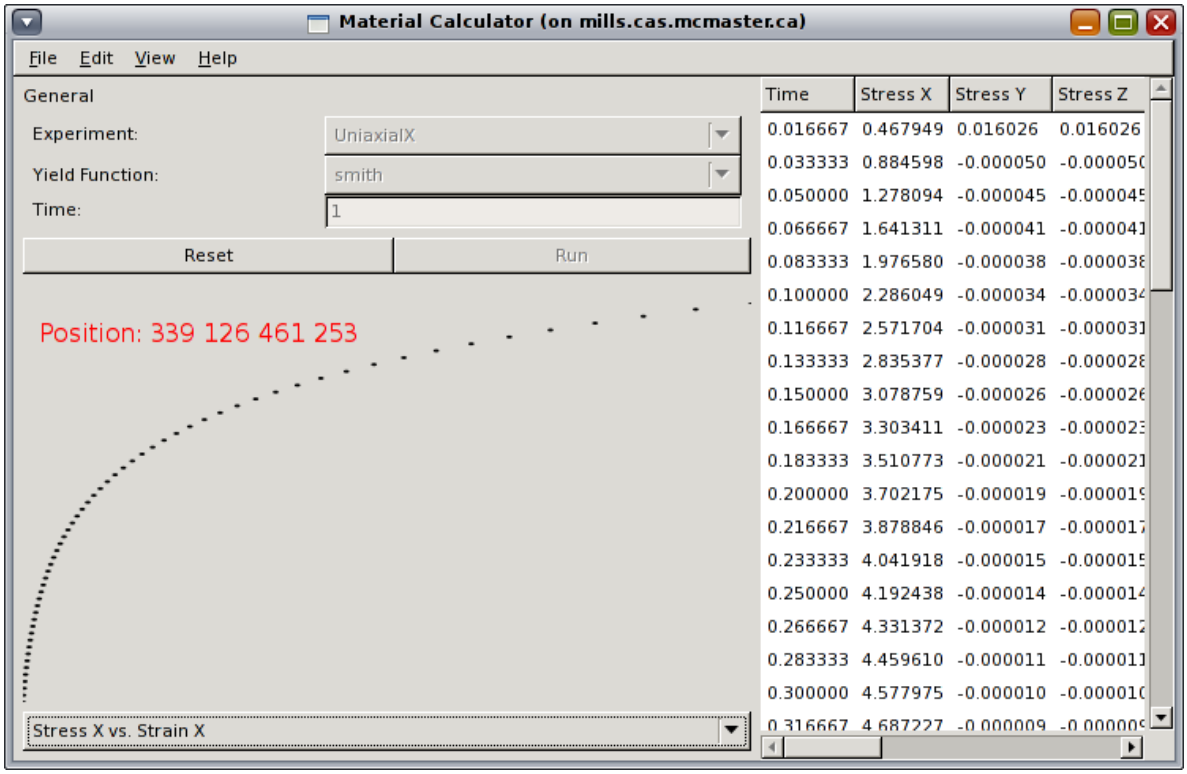


Fig. 8: MatCalc v. 0.1 GUI after RUN

The right panel now displays the data table produced by the simulated test, and the lower part of the left panel shows a graph with the plot of stress values as a function of strain (the user can select the dimension, X, Y or Z, in which to plot the stress vs. strain values from the data table, with the combobox provided beneath the graph panel).

Several usability hindrances can be noted:

- The time step was not a user configurable option, and is fixed at 1/60 s.
- The graph panel textual information labelled 'Position', which is updated as explored with the pointing device, just refers to the pixel range and pointing device pixel coordinates of the graph panel, but has no reference to the underlying plot data (*i.e.* the data table).
- The left and right panels are not resizable to the user's convenience, so that the data table is awkward to visualize even on wide screens.
- The data table does not present the information in scientific format.
- The material test and model constant's values have been occluded by the data table after the simulation run.

The console ouput (`stdout` stream) generated by the program is shown in Fig. 9:

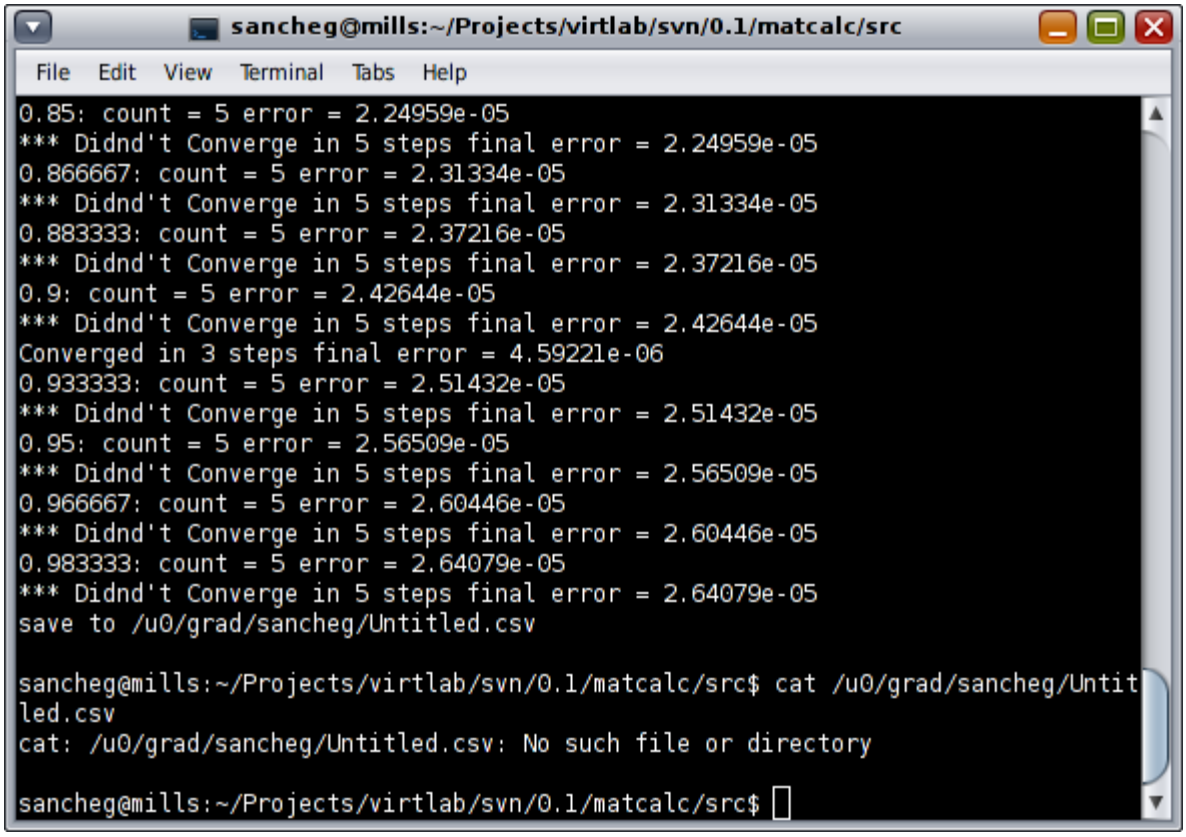

sancheg@mills:~/Projects/virtlab/svn/0.1/matcalc/src
File Edit View Terminal Tabs Help
0.85: count = 5 error = 2.24959e-05
*** Didnd't Converge in 5 steps final error = 2.24959e-05
0.866667: count = 5 error = 2.31334e-05
*** Didnd't Converge in 5 steps final error = 2.31334e-05
0.883333: count = 5 error = 2.37216e-05
*** Didnd't Converge in 5 steps final error = 2.37216e-05
0.9: count = 5 error = 2.42644e-05
*** Didnd't Converge in 5 steps final error = 2.42644e-05
Converged in 3 steps final error = 4.59221e-06
0.933333: count = 5 error = 2.51432e-05
*** Didnd't Converge in 5 steps final error = 2.51432e-05
0.95: count = 5 error = 2.56509e-05
*** Didnd't Converge in 5 steps final error = 2.56509e-05
0.966667: count = 5 error = 2.60446e-05
*** Didnd't Converge in 5 steps final error = 2.60446e-05
0.983333: count = 5 error = 2.64079e-05
*** Didnd't Converge in 5 steps final error = 2.64079e-05
save to /u0/grad/sancheg/Untitled.csv
sancheg@mills:~/Projects/virtlab/svn/0.1/matcalc/src$ cat /u0/grad/sancheg/Untit
led.csv
cat: /u0/grad/sancheg/Untitled.csv: No such file or directory
sancheg@mills:~/Projects/virtlab/svn/0.1/matcalc/src$

Fig. 9: MatCalc v. 0.1 `stdout`

Besides a relatively difficult to read output, the screenshot also shows that the ‘write to file’ option is in fact *not* supported from MatCalc’s v. 0.1 GUI (nor is the ‘read from file’ option).

[5.3.2.2] MatCalc v. 1.1 GUI

Fig. 10 shows the GUI frontend for MatCalc, v. 1.1:

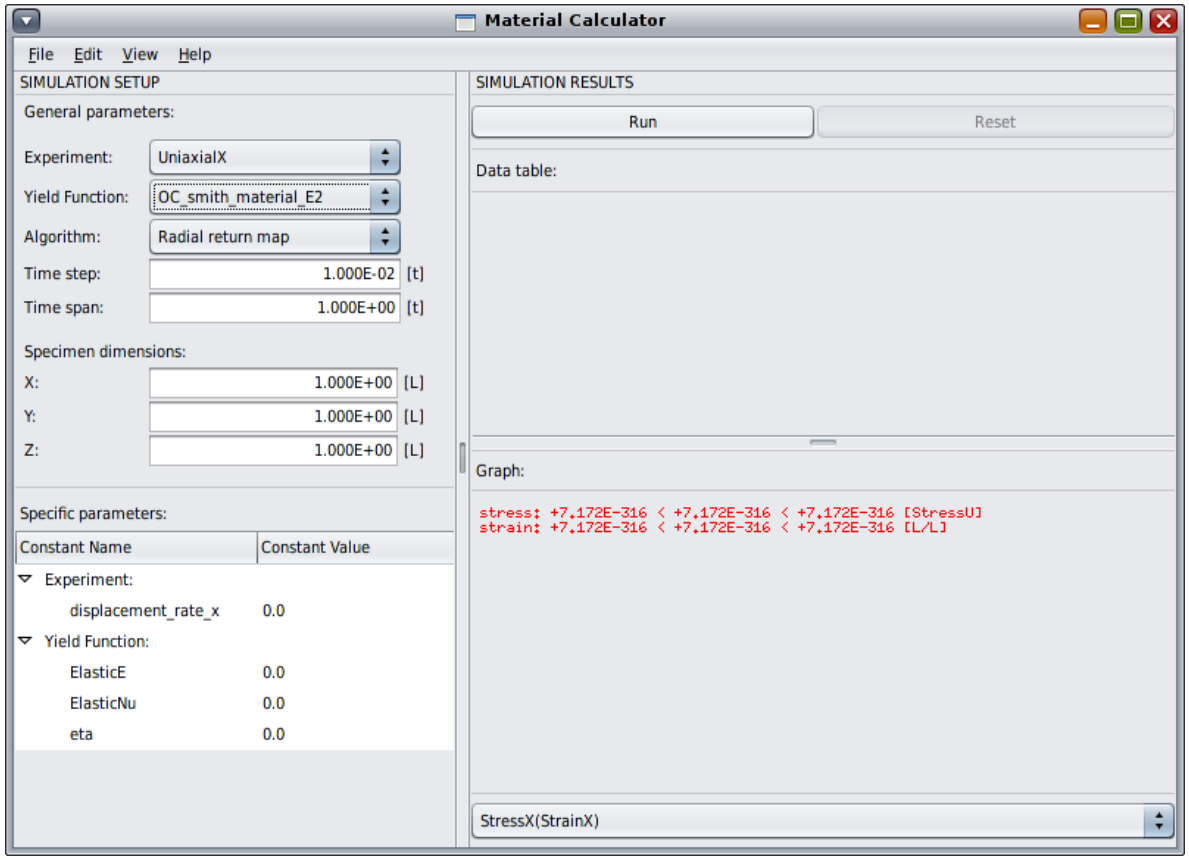


Fig. 10: MatCalc v. 1.1 GUI after RESET

The main usability issues of MatCalc's v. 0.1 GUI were addressed by firstly providing a meaningful and more intuitive overall organization to the interface:

- The left panel is dedicated to the 'SIMULATION SETUP' controls, and has two sections:
  - A 'General parameters' upper section, that allows selection of the material test, material model, and algorithm, and allows the edition of the time step, time span and test specimen dimensions.
  - A 'Specific parameters' lower section, that allows the edition of the specific material test and model parameter values.
- The right panel is dedicated to the 'SIMULATION RESULTS' widgets, and has two sections:
  - A 'Data table' upper section, for the numerical output of the simulation results.
  - A 'Graph' lower section, for the graphical output of the simulation results.

MatCalc v. 1.1 GUI also allows the user further control of the simulation with respect to the original GUI version, by providing edit boxes for setting the time step and test specimen dimensions, and a control for selecting the FEM algorithm variant for the simulation[33].

Fig. 11 shows MatCalc's v. 1.1 GUI after a simulation run:

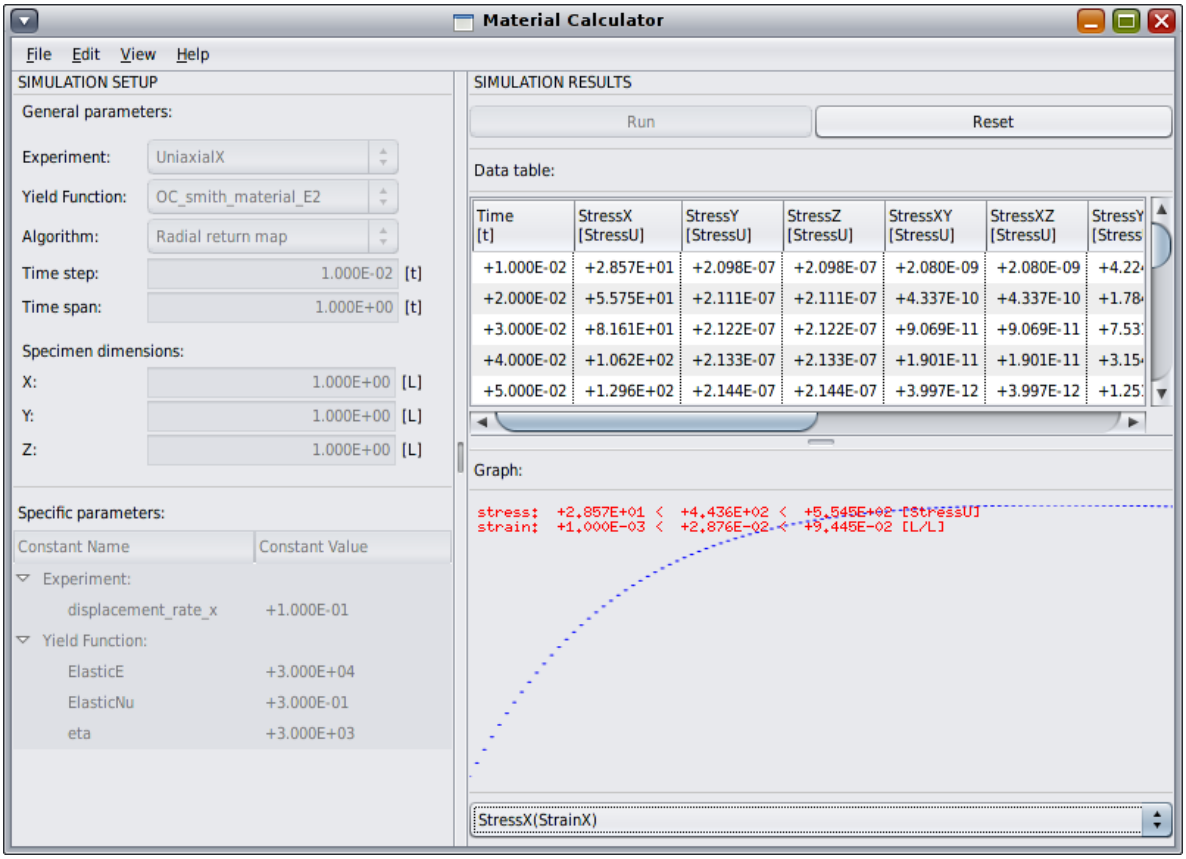


Fig. 11: MatCalc v. 1.1 GUI after RUN

Among further usability improvements in this version of the GUI, it can be noted that:

- The complete simulation setup data is visible against the simulation results.
- The data table uses scientific format for displaying the numerical information, shows the corresponding physical units for each data column, has an easier to read layout by the shading of every other row, and can be scrolled independently of the graph panel.
- The graph panel textual information, updated as explored with the pointing device, shows the 'physical' value and units (as opposed to mere pixel coordinates) within the corresponding axis limits, which enables a direct comparison with the data table above. However, the graph panel is still the best candidate for improvement in MatCalc's GUI; section 5.3.2.3 lists a number of improvements envisioned for this panel.

[33] These features were already available in the underlying Virtlab library, yet not exploited in the original GUI.

Although not visible in the screenshot, pop-up hints were added to each of the widgets in the GUI, that provide the user with additional usage information for each control when he/she hovers each control with the pointing device[34]. Also not visible in the screenshot, but implemented in v. 1.1 of MatCalc's GUI frontend, is the possibility of displaying the shear stresses and strains in graphical format, which significantly improves the ability of the user to explore the simulated material behaviour[35].

In v. 1.1, the right panel can be resized independently of the left panel, which in wide displays allows the user to view the whole width of the data table (Fig. 12):

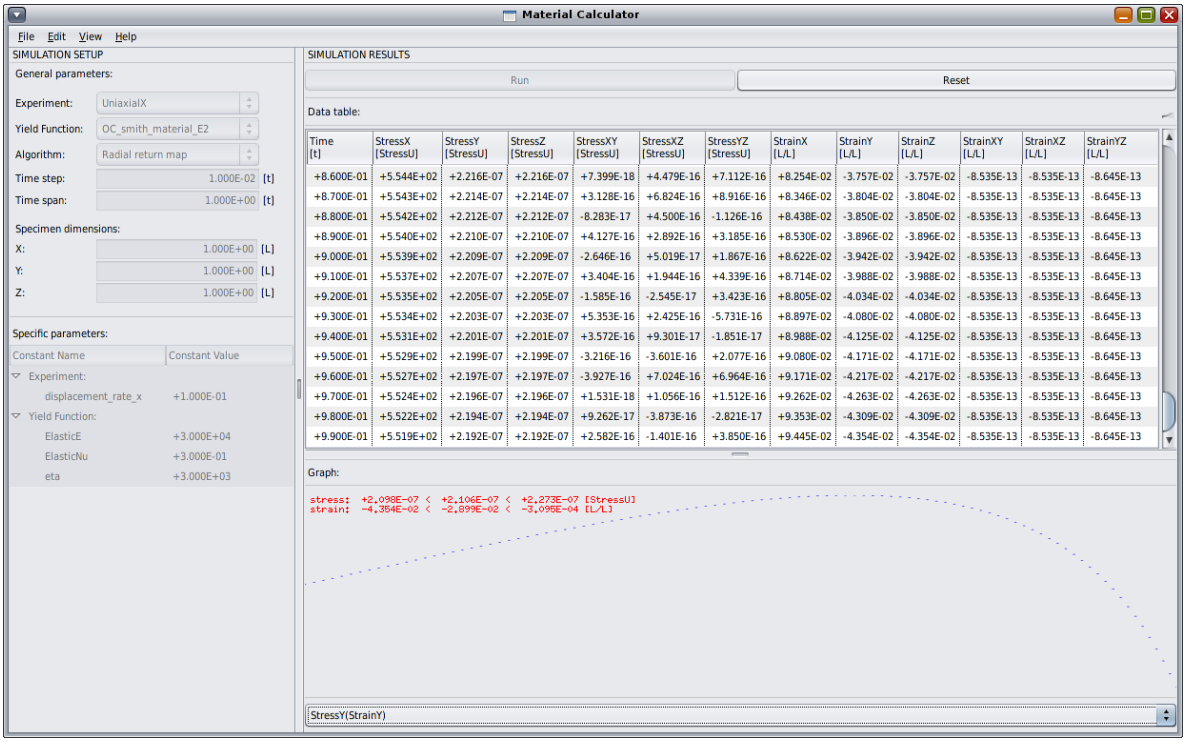


Fig. 12: MatCalc v. 1.1 independent resizing capabilities of left and right panels

The right panel allows the user to resize its upper and lower sections by dragging the horizontal divider between these, which ultimately allows the user to view exclusively the graph or data table, as shown in Fig. 13 and Fig. 14, respectively.

[34] This feature, though, requires the underlying Gtk+ library to provide the corresponding support. Given that is a minor usability issue, which can be offset completely by the user's manual, portability of this feature was not deemed a concern.

[35] These possibility was already provided by the underlying Virtlab library, yet not exploited in the original GUI. Further possibilities, like plotting data against time can be easily implemented, but the information provided in the graph panel would require a more involved code refactoring unless the physical units display is dropped therein (a minor issue, actually, since the physical units can be easily 'hardcoded' into the corresponding graph selection combobox label).

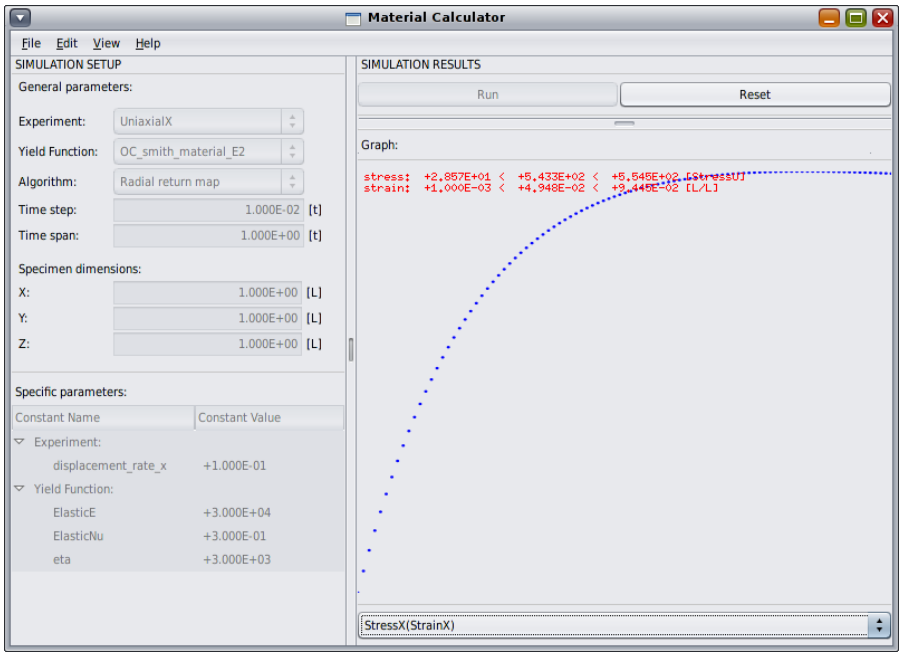


Fig. 13: MatCalc v. 1.1 graph section of the right panel

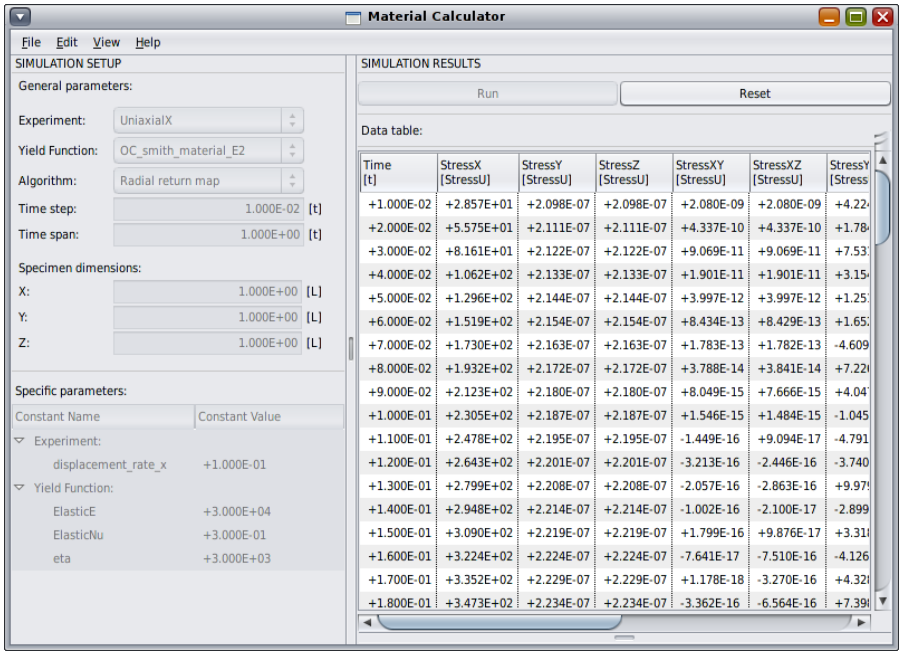


Fig. 14: MatCalc v. 1.1 data table section of the right panel

The aforementioned improvements did not involve significant code refactoring, but mostly understanding the interaction between the GUI driver code and the GUI layout description. A significantly more involved understanding was required, though, to match the underlying library functions to the GUI driver code.

As expected, the insight gained while improving the usability of the GUI frontend helped to implement file input/output functionality. Output files can be used as input *e.g.* for viewing in the GUI. The input file parsing steps are dumped to the console as reference (Fig. 15):

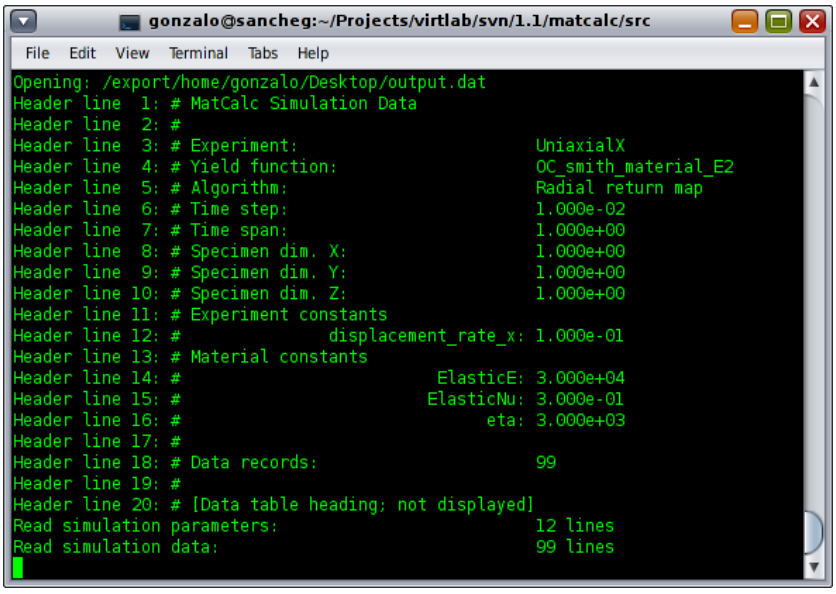

gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src
File Edit View Terminal Tabs Help
Opening: /export/home/gonzalo/Desktop/output.dat
Header line 1: # MatCalc Simulation Data
Header line 2: #
Header line 3: # Experiment: UniaxialX
Header line 4: # Yield function: OC_smith_material_E2
Header line 5: # Algorithm: Radial return map
Header line 6: # Time step: 1.000e-02
Header line 7: # Time span: 1.000e+00
Header line 8: # Specimen dim. X: 1.000e+00
Header line 9: # Specimen dim. Y: 1.000e+00
Header line 10: # Specimen dim. Z: 1.000e+00
Header line 11: # Experiment constants
Header line 12: # displacement_rate_x: 1.000e-01
Header line 13: # Material constants
Header line 14: # ElasticE: 3.000e+04
Header line 15: # ElasticNu: 3.000e-01
Header line 16: # eta: 3.000e+03
Header line 17: #
Header line 18: # Data records: 99
Header line 19: #
Header line 20: # [Data table heading; not displayed]
Read simulation parameters: 12 lines
Read simulation data: 99 lines

Fig. 15: MatcCalc v. 1.1 file input parsing progress shown in `stdout`.

The console output of the software product was improved, for easier readability[36] (Fig. 16):

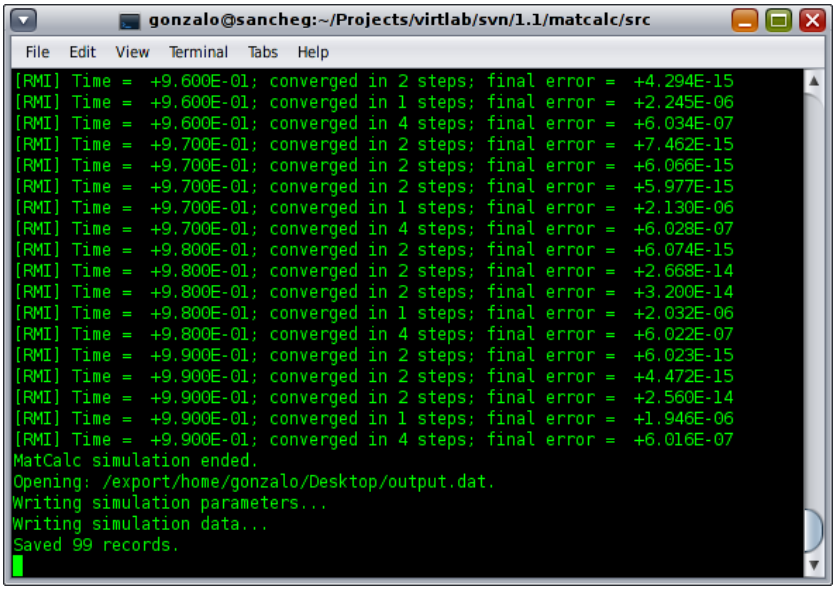

gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src
File Edit View Terminal Tabs Help
[RMI] Time = +9.600E-01; converged in 2 steps; final error = +4.294E-15
[RMI] Time = +9.600E-01; converged in 1 steps; final error = +2.245E-06
[RMI] Time = +9.600E-01; converged in 4 steps; final error = +6.034E-07
[RMI] Time = +9.700E-01; converged in 2 steps; final error = +7.462E-15
[RMI] Time = +9.700E-01; converged in 2 steps; final error = +6.066E-15
[RMI] Time = +9.700E-01; converged in 2 steps; final error = +5.977E-15
[RMI] Time = +9.700E-01; converged in 1 steps; final error = +2.130E-06
[RMI] Time = +9.700E-01; converged in 4 steps; final error = +6.028E-07
[RMI] Time = +9.800E-01; converged in 2 steps; final error = +6.074E-15
[RMI] Time = +9.800E-01; converged in 2 steps; final error = +2.668E-14
[RMI] Time = +9.800E-01; converged in 2 steps; final error = +3.200E-14
[RMI] Time = +9.800E-01; converged in 1 steps; final error = +2.032E-06
[RMI] Time = +9.800E-01; converged in 4 steps; final error = +6.022E-07
[RMI] Time = +9.900E-01; converged in 2 steps; final error = +6.023E-15
[RMI] Time = +9.900E-01; converged in 2 steps; final error = +4.472E-15
[RMI] Time = +9.900E-01; converged in 2 steps; final error = +2.560E-14
[RMI] Time = +9.900E-01; converged in 1 steps; final error = +1.946E-06
[RMI] Time = +9.900E-01; converged in 4 steps; final error = +6.016E-07
MatCalc simulation ended.
Opening: /export/home/gonzalo/Desktop/output.dat.
Writing simulation parameters...
Writing simulation data...
Saved 99 records.

Fig. 16: MatcCalc v. 1.1 information in `stdout` stream, formatted to fit in 80 columns.

[36] This can be easily exploited to generate logs, by simple redirection of the output to a file, eventually processing/filtering the stream by interposing a pipe of commands that accept reading from `stdin` and writing to `stdout` (*e.g.*, `grep`, `awk`, `sed`, `sort`, etc.)

[5.3.2.3] Future work

During the GUI refactoring from v. 0.1 to v. 1.1, several ideas were considered for further enhancing the usability of the software product, in relation to its GUI frontend.

The main issue to improve is to eliminate the relative path reference to the XML definition file for the GUI, which prevents the software to be called from any directory except from that where the XML file is found as `../data/matcalc.glade`. This should be dealt with by means of the 'Autoconf' framework (see section 5.3.1).

Besides the aforementioned issue, the GUI object that offers the greatest potential for improvement is the graphical output display:

- Improve its performance (rendering in slow machines is actually 'painful') or at least add an option to disable updating the 'coordinates' while hovering with the input device over the graph panel, to avoid the constant redrawing/parsing of the data.
- Improve its layout, to avoid the overlap of the plot and the numerical legend; add axis marks, and a grid with major and minor divisions.
- Add option to 'retain' a plot, to enable comparison with successive plots produced by new simulations (*e.g.* for immediate feedback after a parameter value change).
- Add controls for drawing lines between data points, or changing the colour (specially, if plot superposition is implemented).
- Enable plotting every column of the data table against any other (*e.g.* stress or strain values against time)[37].
- Alternatively, or additionally, use an external plotting utility (*e.g.* Gnuplot [WK86]) to manage all graphical output. The linking of this external utilities to Virtlab should be handled by appropriate 'Autoconf' options at compile time, and depending on the availability of the dependencies or specific user choices, might result in that functionality being available or not at run-time (see section 5.3.1, p. 44).

Other possible GUI improvements or fixes are:

- Enable the display of `stdout`/`stderr` streams on a window, for those users who are not aware of, or do not prefer, the use of the console terminal.
- Implement a 'status' bar,
- Implement a 'progress bar' for providing feedback on the simulation running status[38]; this could be included in the 'status' bar previously suggested.
- Implement a 'Stop' button to interrupt the simulation[39].

[37] This can be implemented with relative ease, by providing a combobox for each graph axis, in which the entries would simply correspond to every column of the data table; the underlying rendering does not assume anything except that it is passed two data vectors.

[38] This would require modifying the core FEM algorithms and respective interfaces, so that the 'current simulation time' is accessible from outside of these. A simple `current_time` variable passed by reference to the integration algorithms could be a starting point

[39] This would require modifying the core FEM algorithms and perhaps their respective interfaces, so that these are aware of a 'stop' signal. The main loop in the algorithm could check the 'stop' condition at the same time when it checks if the time span for the simulation is reached. Combined with a general knowledge of the 'current simulation time' this might enable pausing the simulation, or even execute it in stepping mode.

The separation of concerns that was performed 'under the hood' while updating the GUI could be exploited to develop a Java Applet-based GUI, that would enable the operation of the software product from a web browser (see section 5.3.5. p. 76). This would be a significant further usability and portability enhancement for the product, specially if adequately combined with the 'Autoconf' building stages to enable the user the choice of GUI type (Gtk+ or Java Applet) .

[5.3.3] MatCalc's CLI

As part of the sought improvement in the usability and portability qualities, a CLI frontend was developed for MatCalc to enable 'batch' operations.

MatCalc's CLI provides a simple yet informative usage message if invoked with the `-h` switch (Fig. 17):

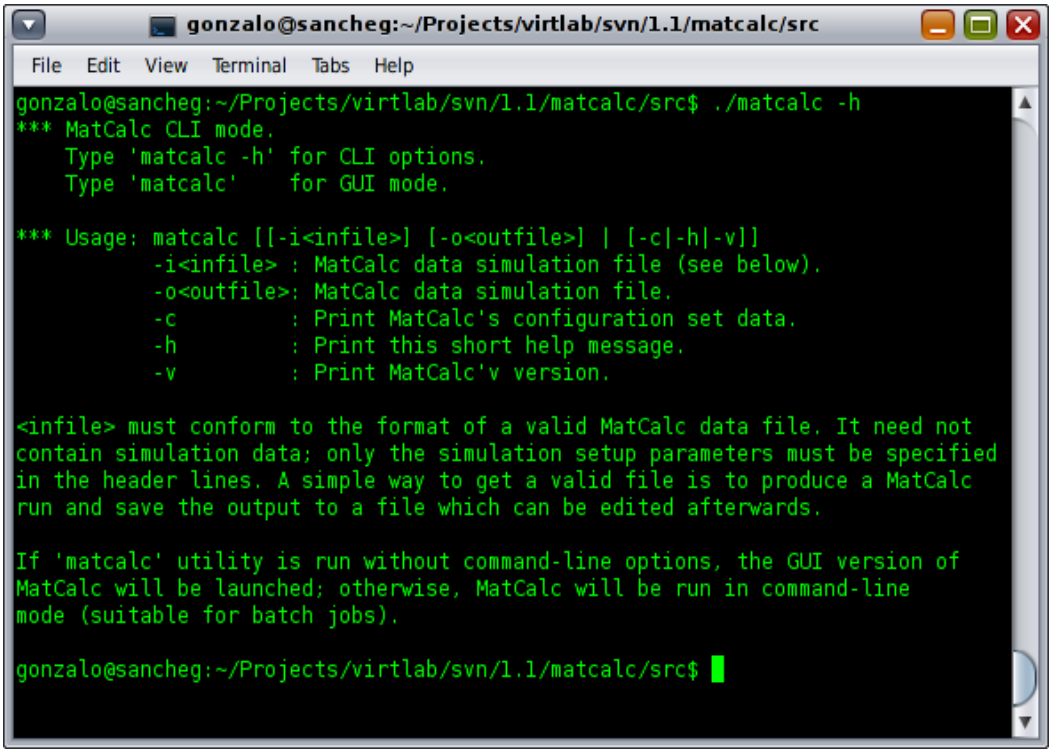

gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src
File Edit View Terminal Tabs Help
gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src$ ./matcalc -h
*** MatCalc CLI mode.
Type 'matcalc -h' for CLI options.
Type 'matcalc' for GUI mode.
*** Usage: matcalc [[-i<infile>] [-o<outfile>] | [-c|-h|-v]]
-i<infile> : MatCalc data simulation file (see below).
-o<outfile>: MatCalc data simulation file.
-c : Print MatCalc's configuration set data.
-h : Print this short help message.
-v : Print MatCalc'v version.
<infile> must conform to the format of a valid MatCalc data file. It need not
contain simulation data; only the simulation setup parameters must be specified
in the header lines. A simple way to get a valid file is to produce a MatCalc
run and save the output to a file which can be edited afterwards.
If 'matcalc' utility is run without command-line options, the GUI version of
MatCalc will be launched; otherwise, MatCalc will be run in command-line
mode (suitable for batch jobs).
gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src$

Fig. 17: MatCalc v. 1.1 CLI usage message

A typical invocation of MatCalc CLI frontend for a 'batch' run would therefore be:

```
user@node:~/path/to/matcalc$ ./matcalc -i/path/to/infile.dat -o/path/to/outfile.dat
```

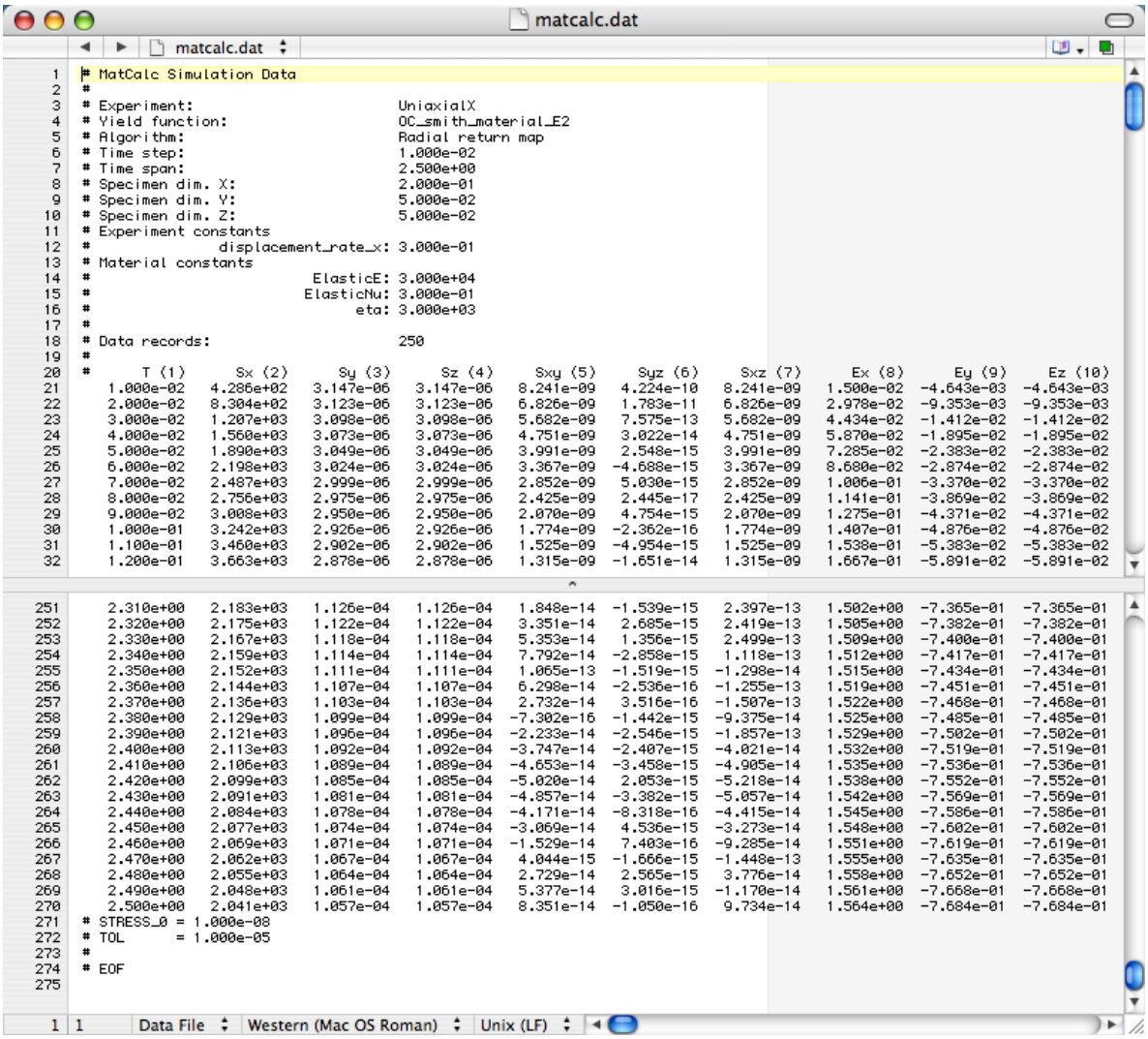
As can be inferred from the sample command line above, or from inspection of the command-line syntax in Fig. 17, MatCalc's CLI does not provide a switch for every parameter or option present in the GUI frontend[40], and unlike the GUI frontend, requires an input file[41].

The development of the CLI frontend relied heavily on the previous implementation of proper file-based input/output functionality in the library.

[40] Material test, material model, integration algorithm, time step and span, test specimen dimensions, material test and model constants.

[41] Among the factors that contributed to the decision of using an input file for the simulation setup in lieu of command-line switches was the number of parameters required for the simulation setup: a command line invocation would result complicated for the user, and from the code's perspective would imply parsing options with variable number of arguments (for the material test and material model constant values).

The file format was designed to accommodate both input and output needs, thus having a 'header' for holding the simulation setup parameter values, and a data section for holding the simulation results. A sample MatCalc input/output setup/data file is shown in Fig. 18:


matcalc.dat

```
# MatCalc Simulation Data
#
# Experiment:                       UniaxialX
# Yield function:                   OC_smith_material_E2
# Algorithm:                        Radial return map
# Time step:                        1.000e-02
# Time span:                        2.500e+00
# Specimen dim. X:                  2.000e-01
# Specimen dim. Y:                  5.000e-02
# Specimen dim. Z:                  5.000e-02
# Experiment constants
#              displacement_rate_x: 3.000e-01
# Material constants
#                         ElasticE: 3.000e+04
#                        ElasticNu: 3.000e-01
#                               eta: 3.000e+03
#
# Data records:                     250
#
#      T (1)      Sx (2)      Sy (3)      Sz (4)      Sxy (5)     Syz (6)     Sxz (7)      Ex (8)      Ey (9)      Ez (10)
   1.000e-02   4.286e+02   3.147e-06   3.147e-06   8.241e-09   4.224e-10   8.241e-09   1.500e-02  -4.643e-03  -4.643e-03
   2.000e-02   8.304e+02   3.123e-06   3.123e-06   6.826e-09   1.783e-11   6.826e-09   2.978e-02  -9.353e-03  -9.353e-03
   3.000e-02   1.207e+03   3.098e-06   3.098e-06   5.682e-09   7.575e-13   5.682e-09   4.434e-02  -1.412e-02  -1.412e-02
   4.000e-02   1.560e+03   3.073e-06   3.073e-06   4.751e-09   3.022e-14   4.751e-09   5.870e-02  -1.895e-02  -1.895e-02
   5.000e-02   1.890e+03   3.049e-06   3.049e-06   3.991e-09   2.548e-15   3.991e-09   7.285e-02  -2.383e-02  -2.383e-02
   6.000e-02   2.198e+03   3.024e-06   3.024e-06   3.367e-09  -4.688e-15   3.367e-09   8.680e-02  -2.874e-02  -2.874e-02
   7.000e-02   2.487e+03   2.999e-06   2.999e-06   2.852e-09   5.030e-15   2.852e-09   1.006e-01  -3.370e-02  -3.370e-02
   8.000e-02   2.756e+03   2.975e-06   2.975e-06   2.425e-09   2.445e-17   2.425e-09   1.141e-01  -3.869e-02  -3.869e-02
   9.000e-02   3.008e+03   2.950e-06   2.950e-06   2.070e-09   4.754e-15   2.070e-09   1.275e-01  -4.371e-02  -4.371e-02
   1.000e-01   3.242e+03   2.926e-06   2.926e-06   1.774e-09  -2.362e-16   1.774e-09   1.407e-01  -4.876e-02  -4.876e-02
   1.100e-01   3.460e+03   2.902e-06   2.902e-06   1.525e-09  -4.954e-15   1.525e-09   1.538e-01  -5.383e-02  -5.383e-02
   1.200e-01   3.663e+03   2.878e-06   2.878e-06   1.315e-09  -1.651e-14   1.315e-09   1.667e-01  -5.891e-02  -5.891e-02

   2.310e+00   2.183e+03   1.126e-04   1.126e-04   1.848e-14  -1.539e-15   2.397e-13   1.502e+00  -7.365e-01  -7.365e-01
   2.320e+00   2.175e+03   1.122e-04   1.122e-04   3.351e-14   2.685e-15   2.419e-13   1.505e+00  -7.382e-01  -7.382e-01
   2.330e+00   2.167e+03   1.118e-04   1.118e-04   5.353e-14   1.356e-15   2.499e-13   1.509e+00  -7.400e-01  -7.400e-01
   2.340e+00   2.159e+03   1.114e-04   1.114e-04   7.792e-14  -2.858e-15   1.118e-13   1.512e+00  -7.417e-01  -7.417e-01
   2.350e+00   2.152e+03   1.111e-04   1.111e-04   1.065e-13  -1.519e-15  -1.298e-14   1.515e+00  -7.434e-01  -7.434e-01
   2.360e+00   2.144e+03   1.107e-04   1.107e-04   6.298e-14  -2.536e-16  -1.255e-13   1.519e+00  -7.451e-01  -7.451e-01
   2.370e+00   2.136e+03   1.103e-04   1.103e-04   2.732e-14   3.516e-16  -1.507e-13   1.522e+00  -7.468e-01  -7.468e-01
   2.380e+00   2.129e+03   1.099e-04   1.099e-04  -7.302e-16  -1.442e-15  -9.375e-14   1.525e+00  -7.485e-01  -7.485e-01
   2.390e+00   2.121e+03   1.096e-04   1.096e-04  -2.233e-14  -2.546e-15  -1.857e-13   1.529e+00  -7.502e-01  -7.502e-01
   2.400e+00   2.113e+03   1.092e-04   1.092e-04  -3.747e-14  -2.407e-15  -4.021e-14   1.532e+00  -7.519e-01  -7.519e-01
   2.410e+00   2.106e+03   1.089e-04   1.089e-04  -4.653e-14  -3.458e-15  -4.905e-14   1.535e+00  -7.536e-01  -7.536e-01
   2.420e+00   2.099e+03   1.085e-04   1.085e-04  -5.020e-14   2.053e-15  -5.218e-14   1.538e+00  -7.552e-01  -7.552e-01
   2.430e+00   2.091e+03   1.081e-04   1.081e-04  -4.857e-14  -3.382e-15  -5.057e-14   1.542e+00  -7.569e-01  -7.569e-01
   2.440e+00   2.084e+03   1.078e-04   1.078e-04  -4.171e-14  -8.318e-16  -4.415e-14   1.545e+00  -7.586e-01  -7.586e-01
   2.450e+00   2.077e+03   1.074e-04   1.074e-04  -3.069e-14   4.536e-15  -3.273e-14   1.548e+00  -7.602e-01  -7.602e-01
   2.460e+00   2.069e+03   1.071e-04   1.071e-04  -1.529e-14   7.403e-16  -9.285e-14   1.551e+00  -7.619e-01  -7.619e-01
   2.470e+00   2.062e+03   1.067e-04   1.067e-04   4.044e-15  -1.666e-15  -1.448e-13   1.555e+00  -7.635e-01  -7.635e-01
   2.480e+00   2.055e+03   1.064e-04   1.064e-04   2.729e-14   2.565e-15   3.776e-14   1.558e+00  -7.652e-01  -7.652e-01
   2.490e+00   2.048e+03   1.061e-04   1.061e-04   5.377e-14   3.016e-15  -1.170e-14   1.561e+00  -7.668e-01  -7.668e-01
   2.500e+00   2.041e+03   1.057e-04   1.057e-04   8.351e-14  -1.050e-16   9.734e-14   1.564e+00  -7.684e-01  -7.684e-01
# STRESS_0 = 1.000e-08
# TOL      = 1.000e-05
#
# EOF
```

Data File | Western (Mac OS Roman) | Unix (LF)

Fig. 18: MatCalc v. 1.1 input/output file sample

The rationale behind the adoption of the above format is based on the following ideas:

- A single file containing both the simulation setup with the simulation results is essential for proper future reference. Hence – if unmodified – it is particularly suited for loading into MatCalc's GUI for display, since it ensures consistency between the displayed simulation setup and results.
- A plain ASCII text file is almost universally portable, and ensures readability by human readers with a simple text editor. It is particularly suitable for analysis with software tools like Octave [Eat94] computing environments, and plotting utilities such as Gnuplot [WK86].

The column and comment format were in fact chosen for compatibility with the cross-platform Gnuplot plotting utility [WK86]. An example plot, produced by the Gnuplot script in Fig. 19, is displayed on Fig. 20 (where the interactive measurement facilities provided by Gnuplot is used):

matcalc-1.1_gnuplot.plt <gonzalo>

File Edit Search Preferences Shell Macro Windows Help

```
1  set style data lines
2
3  set border 31 lw 0.75
4
5  set format x "%1.1e"
6  set format y "%1.1e"
7
8  set mxtics
9  set mytics
10 set grid
11
12 set title  "MatCalc v. 1.1 simulation results"
13 set xlabel "strain [L/L]"
14 set ylabel "stress [StressU]"
15 set key left top samplen 1
16
17 plot   "output.dat" using  8:2 with lp lw 1 pt 1 ps 1 title "  X: E2 - Uniaxial_X"
18
19 #plot   "output.dat" using  9:3 with lp lw 1 pt 2 ps 1 title "  Y: E2 - Uniaxial_X"
20 #replot "output.dat" using 10:4 with lp lw 1 pt 3 ps 1 title "  Z: E2 - Uniaxial_X"
21
22 #plot   "output.dat" using 11:5 with lp lw 1 pt 4 ps 1 title " XY: E2 - Uniaxial_X"
23 #replot "output.dat" using 12:6 with lp lw 1 pt 5 ps 1 title " YZ: E2 - Uniaxial_X"
24 #replot "output.dat" using 13:7 with lp lw 1 pt 6 ps 1 title " XZ: E2 - Uniaxial_X"
25
26 unset key
27 unset ylabel
28 unset xlabel
29 unset title
30 unset grid
31 unset mytics
```

Fig. 19: Sample Gnuplot script for plotting data from a MatCalc v. 1.1 data file.

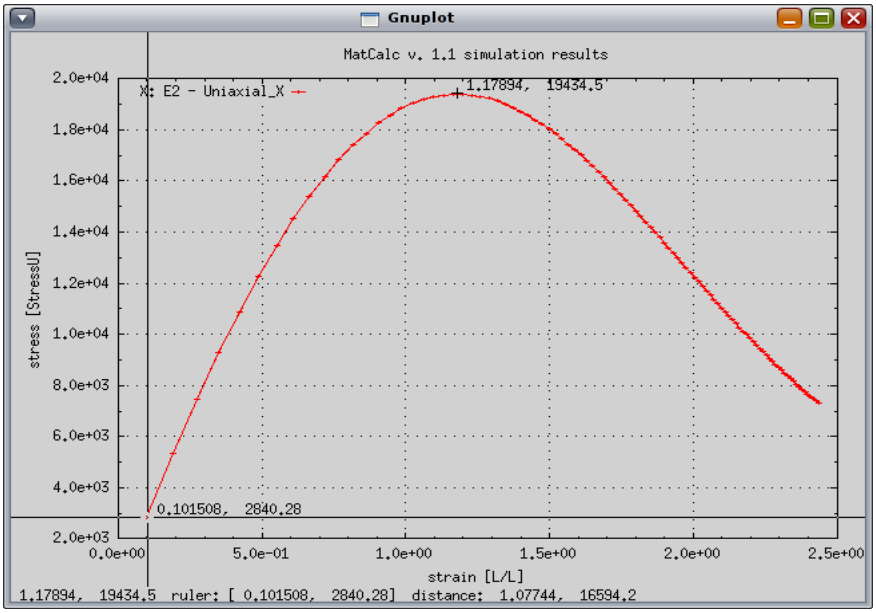


Fig. 20: Gnuplot graph (data: line 17 in Fig. 19) showing the use of interactive measurements.

Any default values assumed by MatCalc are displayed by invoking it with the switch `-c` (Fig. 21):

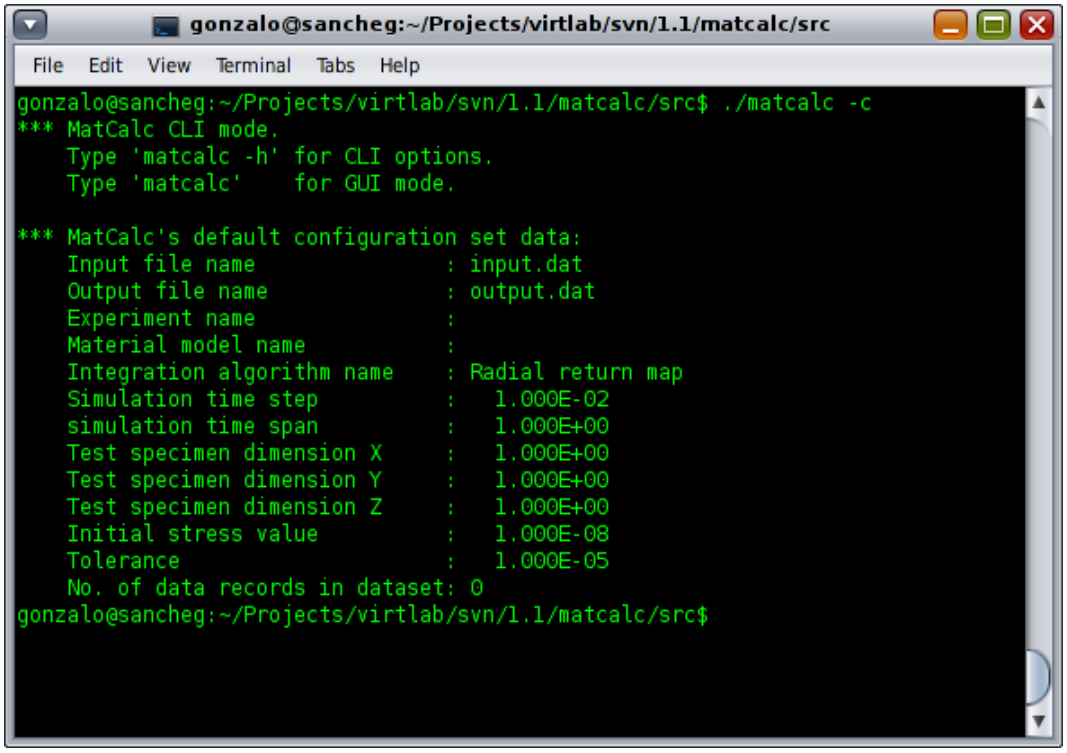

gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src
File Edit View Terminal Tabs Help
gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src$ ./matcalc -c
*** MatCalc CLI mode.
Type 'matcalc -h' for CLI options.
Type 'matcalc' for GUI mode.
*** MatCalc's default configuration set data:
Input file name : input.dat
Output file name : output.dat
Experiment name :
Material model name :
Integration algorithm name : Radial return map
Simulation time step : 1.000E-02
simulation time span : 1.000E+00
Test specimen dimension X : 1.000E+00
Test specimen dimension Y : 1.000E+00
Test specimen dimension Z : 1.000E+00
Initial stress value : 1.000E-08
Tolerance : 1.000E-05
No. of data records in dataset: 0
gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src$

Fig. 21: MatCalc v. 1.1 CLI configuration printout

The information provided with the `-h` and `-c` switches is intended to enable the user to quickly run and 'try' the software – a basic usability feature, often overlooked. To that same purpose, sample input and output files named `input.dat` and `output.dat` are included in the software package.

The console output produced by MatCalc's CLI frontend at run-time – except for the usage message and configuration information shown in figures 17 (p. 69) and 21 (p. 72) – is exactly the same as that produced by MatCalc's GUI frontend at run-time; please refer to figures 15 (p. 66) and 16 (p. 66) for an example. This commonality stems from the fact that the underlying library usage is exactly the same for both CLI and GUI frontends, and is a result of a design decision to address the maintainability, understandability, verifiability, and reliability qualities (see section 5.3.5, Fig. 22, p. 80).

[5.3.4] MatGen

Except for slight updates in the building process (see section 5.3.1, p. 44) and the documentation improvements (see section 5.3.7, p. 91) no specific work was performed on MatGen modules.

The reason for the scarce improvements to MatGen specific modules reside in two facts:

1. Virtlab Project goals implied focusing mainly on MatCalc modules
2. MatGen's basic purpose, the material model classes code generation, produced inconsistent results:
   - sometimes no code is generated
   - sometimes code is generated that would not compile
   - sometimes code is generated that would successfully compile, but would produce *wrong results* when run within MatGen.

A significant effort was effected towards diagnosing and fixing the problem, but unfortunately, without success. This problem was put aside in the face of the other priorities of the Virtlab project, and is now the first one indicated as future work on Virtlab Project (see section 7.2, p. 139).

[5.3.4.1] Code generation statement test

A tool, `test_EvalMapleStatement`, was developed with the intention to diagnose and fix the code generation problem detected with MatGen.

> Section 9.11 (p. 188) lists the contents of the `test_EvalMapleStatement.cc` file.

The tool uses the existing MatGen modules to interface the Maple computer algebra system, in the same way MatGen does for the purposes of generating the code for the material model classes.

The tool is designed to simply pass Maple an expression – assumed to be a valid expression for the Maple system – and return the output that Maple produces after evaluating it:

```
sancheg@mills:~$ ./test_EvalMapleStatement
Usage: test_EvalMapleStatement "<expression>"
where <expression> is a Maple expression terminated with ';'
e.g.:
   "CodeGeneration[C](<expression>,resultname=<var_name>);"
   "1.0/3.0;"
```

As suggested in the above 'Usage' message, the example expression `"1.0/3.0;"` is given as argument to the tool so that it forwards the expression to Maple for evaluation. This returns the expected numerical answer:

```
sancheg@mills:~$ ./test_EvalMapleStatement "1.0/3.0;"
*** Creating new mapleclient instance...
```

```
*** Setting mapleclient output to stdout...
*** Starting Maple...
true
*** Resetting Maple...
false
*** Evaluating expression: 1.0/3.0;
.3333333333
*** Stopping Maple...
*** Freeing allocated memory...
```

When the tool is instructed to submit Maple an expression for generating C code as the statement to evaluate, the returned answer is again within the expected answers Maple could provide:

```
sancheg@mills:~$ ./test_EvalMapleStatement "CodeGeneration[C](2+3*x^5,resultname=y);"
*** Creating new mapleclient instance...
*** Setting mapleclient output to stdout...
*** Starting Maple...
true
*** Resetting Maple...
false
*** Evaluating expression: CodeGeneration[C](2+3*x^5,resultname=y);
y = 0.2e1 + 0.3e1 * pow(x, 0.5e1);
*** Stopping Maple...
*** Freeing allocated memory...
```

However, when MatGen is used instead of the tool, the inconsistent results listed at the start of section 5.3.4 would be observed.

MatGen was tested in two different platforms, Linux/x86_64 and MacOSX/x86_32, with two different versions of Maple; versions 11 and 13 respectively:

- On the Linux platform, MatGen would initially fail to communicate with Maple; after this issue − related to system setup − was resolved (see section 6.5, p. 130) the code generation would fail to produce code, with the result that MatCalc's build would break when the offending material classes were tried to be compiled.
- On the MacOS X platform, code generation produces files that are an order of magnitude bigger than the originally generated code − MatCalc would require 13 hours of wall-clock time to build − and which would produce wrong results.

The latest tests performed involved a linear viscoelastic material model, in one instance using the originally generated code, and in the other instance using code generated recently in the MacOS X platform. The `dset_diff` tool (see section 5.3.8.2, p. 96) was used to compare the modelled material behaviour results, simulated with MatCalc for each instance. The tests revealed the following:

- The simulation results are *completely different* for simulations computed with the *Radial Return map* or *Viscoplastic* integration algorithms (the relative error between results reaches 100% for some of the data columns in the datasets compared).
- The simulation results are *identical* for simulations computed with the *Elastic* integration algorithm (the relative error between results is 0% in all data vectors compared).

[5.3.4.2] Future work

Future work in MatGen should cover the following issues:

- Diagnose and fix the code generation problems.
- Eliminate the relative path reference to the XML definition file for the GUI, which prevents the software to be called from any directory except from that where the XML file is found as `../data/yieldf.glade`. This should be dealt with by means of the 'Autoconf' framework (see section 5.3.1).

Additional enhancements could be similar to those stated for MatCalc (see section 5.3.2.3, p. 67):

- Enable the display of `stdout`/`stderr` streams on a window, for those users who are not aware of, or do not prefer, the use of the console terminal.
- Implement a 'status' bar.
- Implement a 'progress bar' for providing feedback on the code generation status; this could be included in the 'status' bar previously suggested.

[5.3.5] Separation of Concerns and Information Hiding

"Basic principles play a key role in handling the difficult problems that arise in multi-person/multi-version programming. The most important principle in software engineering – and in problem-solving generally – is ***separation of concerns*** *[2]. **A problem that is too complex to be solved directly is decomposed into subproblems**. **Subproblems that are still too difficult to solve directly are further decomposed**. **The decomposition is most useful if the subproblems are independent or nearly so**. Thus, considerable effort in software engineering is devoted to (1) the search for decompositions that maximize the independence of the subproblems and (2) careful documentation of the dependencies that remains."*[42] [43] ([HS99], s. 1.2, pp. 3-4)

*"..**information hiding is the principle of segregation of design decisions in a computer program that are most likely to change, thus protecting other parts of the program from extensive modification if the design decision is changed**. The protection involves providing a stable interface which protects the remainder of the program from the implementation (the details that are most likely to change)."*

"***A software module hides information by encapsulating the information into a module or other construct which presents an interface***."

"*A common use of information hiding is to hide the physical storage layout for data so that if it is changed, the change is restricted to a small subset of the total program. For example, if a three-dimensional point (x,y,z) is represented in a program with three floating point scalar variables and later, the representation is changed to a single array variable of size three, a module designed with information hiding in mind would protect the remainder of the program from such a change.*"

http://en.wikipedia.org/wiki/Information_hiding

The modifications and additions introduced in the GUI frontend to MatCalc, and the development of the CLI frontend, were effected with the aforementioned *separation of concerns* and *information hiding* concepts in mind, and were feasible because the underlying library had also been built upon the same principles.

[5.3.5.1] File Input and Output modules

The *file input* and *file output* tasks were assigned to specific modules, which now hide the complexity of populating or querying the original data structure defined in the library, the transference of information between this and the input/output files, and the file 'format' and

[42] **[2]** E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[43] The boldface type is used here to confer the emphasis given in the original text, which used italicized typeface.

'parsing' tasks. The original codebase included code for file input/output[44], but was embedded in the '`dataset`' module that holds the main data structure for the simulation results. Given that storing/retrieving the simulation data from/to a file requires access to the said data structure, having this code within the same module provided some simplifications. However, addition of any data structures in a foreign module – such as the added '`configset`' module, described below – strongly suggested separating the concerns for the sake of understandability and maintainability. Several of the tools the original developer had devised for testing specific aspects of the application had to be refactored to make use the new modules, but overall, the easiness with which these changes could be done, and the understandability gained, made the extra work worthwhile.

[5.3.5.2] GUI and CLI modules

Another aspect of the separations of concerns achieved is that the GUI is now almost completely detached from the library's core, as is the newly developed CLI frontend. This should enable a developer to completely substitute the Gtk+ based GUI for another GUI frontend based on another widget toolkit, or eventually, producing a GUI oriented to web operation, as well as compile Virtlab without GUI support (see sections 5.3.1 and 5.3.2).

[5.3.5.3] `stdout` and `stderr` handling

Part of the code refactoring effected with separation of concerns and information hiding in mind, was to consistently enforce in the codebase the use of functions already present in the library to manage `stdout` and `stderr` streams: all console output is now performed through functions `du_message()` and `du_error()` defined in the `debug_utils` module. This enables to change the behaviour of the console output in a single module in the library, instead of having to deal with `sprintf()` functions throughout the whole codebase, for example, to eliminate all informational messages altogether if performance requirements dictate the need[45].

[5.3.5.4] `configset` module

A `configset` module was introduced with the purpose of handling the run-time configuration of MatCalc. This module is in charge of parsing the command-line options, and provides access functions to create and delete a simple structure in which it keeps information such as the input and output filenames, selected material test and model names, selected integration algorithm, and the values for the time step and span, test specimen dimensions, etc. In the future, this module could support command-line switches for all the possible options, and eventually query environment variables, in which the user might specify default values for the structure elements. This would enable the following three-level priority configuration scheme:

- The application provides built-in defaults for each of the configuration parameters, on which to fall back if any of them is not (properly) set either in the input file, or by the environment variables, or by the command-line switches;

[44] It was not 'hooked' to the existing GUI, and hence, only available for a developer whom used the codebase as a library.

[45] Leaving the `du_message()` function as a 'stub', and requesting the compiler to 'inline' functions should achieve this.

- The values of the configuration parameters set in the input file are overridden by either setting the corresponding environment variables or command-line switches to other (valid) values;
- The values of the configuration parameters set by the environment variables are overridden by setting the corresponding command-line switches to any other (valid) values.

Therefore, the input files could operate as a ‘user templates’ for the application. The environment variables can be used as ‘session defaults’ for the application, which would override any corresponding input files settings. The command-line switches would allow for ultimate tweaks to the configuration at invocation time: for example, the user might use a (single) input file, eventually overriding the test specimen dimensions therein via appropriate environment variables, and then run the simulation several times just changing the time-step in each simulation with an appropriate command-line switch, without the need of editing the input file nor environment variable for just that simple change. This scheme would provide similar flexibility to the CLI frontend as the GUI frontend currently has for ‘experimenting’.

[5.3.5.5] Virtlab v. 1.1 restructure overview

The separation of concerns thus introduced – basically with the file input and output modules, and the configuration module – compounded with the underlying separation of concerns and information hiding policies already implemented in the codebase, allowed the quick development of a material model parameters fitting tool (described in section 5.3.6). This tool provides a clear example of the advantages – in terms of reusability, maintainability and understandability, and indirectly, in terms of reliability and verifiability – gained by a proper modularization and application of information hiding techniques.

Fig. 22 in p. 80 provides an overview of the current relationship between some of the the aforementioned modules in the library, in terms of the files that implement them. This figure does not depict the complete set of files that comprise MatCalc[46] (most notably, the complex interaction between C++ and Ruby is not even hierarchically presented) but focuses on those files that were involved/added to provide the separation of concerns reported in this section.

> **Note**: The pseudo-code snippets that can be seen within each file icon in Fig. 22 (p. 80) actually represent the *key* aspects of the corresponding file's code or purpose thereof – they try to illustrate the access functions or class methods offered, or the logic/control flow – and are intended to provide a rough idea of the application's organization, specially of its 'outer layers'.

[5.3.5.6] Future work

Further work can still be done in specific aspects of the code, particularly, where material test and material model aspects need to be handled. Currently, the `gui`, `matcalc_driver` and `output` modules need to extract the name or constant values of the material virtual tests (experiments) and/or of the material models, for which a series of steps are required (creating a test or model list, querying it in different ways depending on the information at hand, then deleting the list and intermediate objects). It is easy to overlook an object deletion, thus creating memory leaks in the application, so a solution would be implementing specific access functions for each of this purposes, that would further ease the development of new tools and tests for the application, while ensuring the overall reliability. This seemingly minor issue is still a further step in achieving better separation of concerns within the library.

[46] Recall that all work was performed with MatCalc simulation engine in mind, and did *not* address MatGen code generation engine.

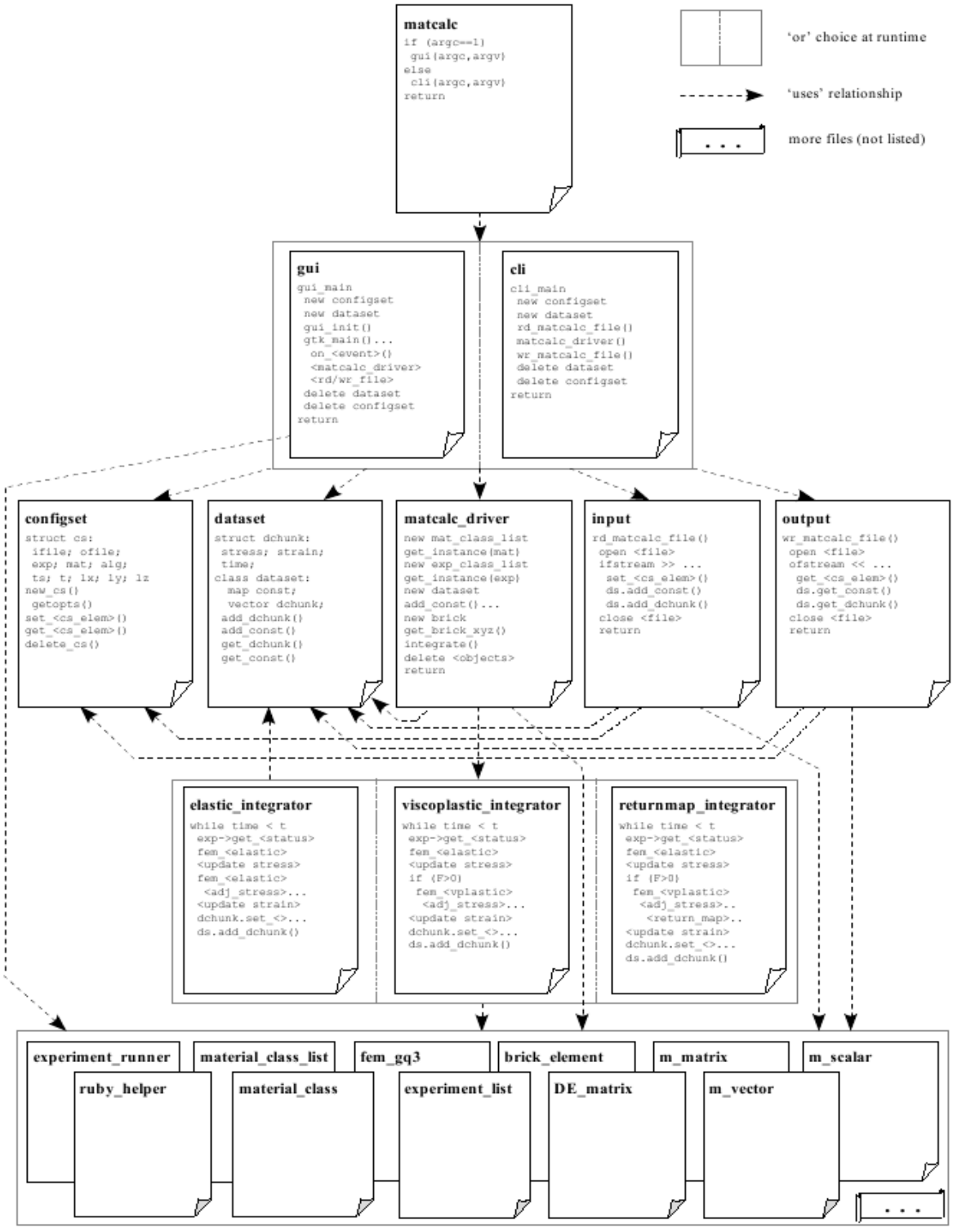


Fig. 22: Virtlab files relationship overview, for the files affected by the separation of concerns applied to the codebase.

[5.3.6] Material Model Parameters Fitting Tool

Towards the goal of testing and improving the software's package accuracy, reliability and verifiability qualities, a tool was developed to fit the material model parameters against a real world material test.

> **Note**: The work on this tool was encompassed within the CES*701 2009 course at CES Department (McMaster University) "*Foundations of Modern Scientific Programming*". The current section contents are almost entirely based on the project report for the referred course [San09].

[5.3.6.1] Tool overview

The purpose of a material model parameters fitting tool for MatCalc is to take data from a real world material test, and data produced from a simulation in MatCalc (for the corresponding material model and virtual test) and return the material model parameter values, fitted as per the real world experimental data. This should allow the users to 'fit' their material models for use within MatCalc to real materials, after which the accuracy, reliability (and ultimately, the verifiability) of Virtlab could be checked.

An example of material model implemented by the current MatCalc version is given by the following expression, where σ denotes the material stress, ϵ denotes the material strain, and λ and η are the material model parameters:

$$\sigma + \lambda \dot{\sigma} = 2 \eta \dot{\epsilon}$$

Eq. 7: Example material model

The desired model fitting module would seek to adjust the material model parameters (λ and η for the example given by Eq. 7) by comparing the simulation results produced with 'approximate' values (initial guesses) for those parameters, with the data produced by a real world material test. For that purpose, an appropriate 'cost' function of the material model parameters has to be provided, so that minimizing this objective function returns an 'optimal' set of values for the parameters. A naïve objective function can be the traditional 'least squares' fit, which for the material model example given by Eq. 7 would be:

$$\min_{\vec{p}} \; f(\vec{p}) = \sum_{j=1}^{n} (\sigma_j^{virtual} - \sigma_j^{real})^2, \qquad \vec{p} = (\lambda, \eta)$$

Eq. 8: 'Naïve' objective function for the material model of Eq. 7

To avoid that the optimization method adjusts the parameters λ and η taking into account particularities of either the simulation run or real material test data (*i.e.* artifacts of the simulation and/or real world experiment runs, but *not* of the underlying physical model/reality) the objective function can be extended to include *several* pairs of simulations/tests[47]:

$$\min_{\vec{p}} \; f(\vec{p}) = \sum_{i=1}^{m_{runs}} \sum_{j=1}^{n_{data}} (\sigma_j^{virtual_i} - \sigma_j^{real_i})^2, \qquad \vec{p} = (\lambda, \eta)$$

Eq. 9: Objective function for the material model of Eq. 7

[47] That is, different tests (and corresponding simulations) for the *same* material model – the model parameters have to satisfy *all* tests.

Given the current state of the Virtual Material Laboratory Project software library, the model fitting algorithm to be developed cannot access the simulation internal workings, but only the simulation *results* that are provided either in a data structure in memory or in a text file like the one depicted in Fig. 18 (section 5.3.3, p. 70).

This 'black box' situation impedes the use of any method that requires the use of *e.g.* derivatives of the objective/cost function to be minimized, so a workaround is to use a *direct search* method to fit the experimental data to the simulation results.

[5.3.6.2] Hooke & Jeeves Pattern Search method overview

One popular direct search method is the Hooke and Jeeves *pattern move* or *pattern search* algorithm (1975). A detailed description of the method, alongside an application example, can be found in [SHL83], pp. 180-185. As described therein, given an objective function, the pattern search algorithm proposed by Hooke and Jeeves:

> *"...establishes a search direction by setting a series of position vectors* $\boldsymbol{b}_r$*, called base points, which tend to follow a principal axis (of the objective function). Each move involves an extrapolation of any success achieved by moving from the previous base point to the current one, followed by an exploration of the surface around the extrapolated point to provide corrections for local conditions. The algorithm thus employs a double strategy and the rth iteration comprises:*
>
> *(i) a* ***pattern move*** *from base point* $\boldsymbol{b}_{r-1}$ *to a new trial centre* $\boldsymbol{c}_r$
>
> *(ii) a* ***local exploration*** *around the trial centre* $\boldsymbol{c}_r$ *as a result of which a new basepoint* $\boldsymbol{b}_r$ *is established*
>
> *These position vectors may be defined as*
>
> $$\boldsymbol{b}_r = [(b_1)_r, \ldots, (b_n)_r]^T$$
>
> $$\boldsymbol{c}_r = [(c_1)_r, \ldots, (c_n)_r]^T$$
>
> *in which*
>
> $(b_i)_r$ *= the value of the ith variable at the rth basepoint*
>
> $(c_i)_r$ *= the value of the ith variable at the rth trial centre*
>
> *..."* ([SHL83], pp. 180-181).

In the aforementioned description, the vectors $\boldsymbol{b}_r$ and $\boldsymbol{c}_r$ are simply appropriate names for the parameters vector ($\boldsymbol{p} = (\lambda,\eta)$ in the example) at the *r*th iteration of the algorithm, that only describe the stage of the strategy within the iteration step.

This implies that for *each* iteration of the Hooke and Jeeves method, *several* simulations have to be run to evaluate the objective function *f* given by Eq. 8 or Eq. 9, and after each iteration, the updated parameter values in $\boldsymbol{b}_r$ have to be provided as initial values for the step *r+1*.

A sample search path produced by the Hooke and Jeeves algorithm is depicted in Fig. 23; the table in Fig. 24 gives detailed indications of the actions taken at each step, and why. In the graph legend, 'Success' indicates the cost function evaluation at the point returns a lower value than the current minimum; 'Failure', otherwise.

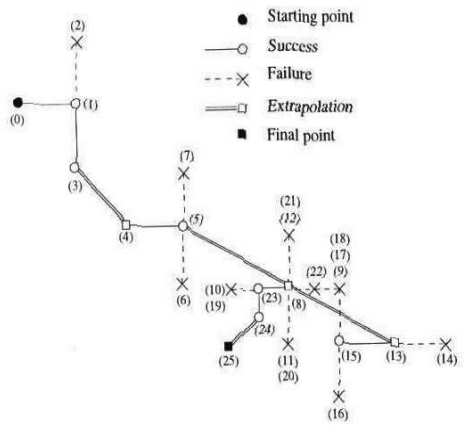


Fig. 23: Sample search by Hooke and Jeeves *pattern move* algorithm. The table in Fig. 24 describes in detail the complete process.

Numbering	Iteration index k	Direction index i	Variable values x_1	x_2	Comparison point	Step lengths s_1	s_2	Remarks
(0)	0	0	0	9	-	2	2	starting point
(1)	0	1	2	9	(0)			success
(2)	0	2	2	11	(1)			failure
(3)	0	2	2	7	(1)			success
(4)	1	0	4	5	-	2	−2	extrapolation
(5)	1	1	6	5	(4),(3)			success, success
(6)	1	2	6	3	(5)			failure
(7)	1	2	6	7	(5)			failure
(8)	2	0	10	3	-,(5)	2	−2	extrapolation, success
(9)	2	1	12	3	(8)			failure
(10)	2	1	8	3	(8)			failure
(11)	2	2	10	1	(8)			failure
(12)	2	2	10	5	(8)			failure
(13)	3	0	14	1	-	2	−2	extrapolation
(14)	3	1	16	1	(13)			failure
(15)	3	1	12	1	(13),(8)			success, failure
(16)	3	2	12	−1	(15)			failure
(17)	3	2	12	3	(15)			failure
(8)	4	0	10	3	-	2	−2	return
(18)	4	1	12	3	(8)			failure
(19)	4	1	8	3	(8)			failure
(20)	4	2	10	1	(8)			failure
(21)	4	2	10	5	(8)			failure
(8)	5	0	10	3	-	1	−1	step lengths halved
(22)	5	1	11	3	(8)			failure
(23)	5	1	9	3	(8)			success
(24)	5	2	9	2	(23),(8)			success, success
(25)	6	0	8	1	-	−1	−1	extrapolation

Fig. 24: Tabular description of the sample search path in Fig. 23.

[5.3.6.3] Design decisions and implementation overview

The implementation of the parameter fitting tool for MatCalc used as basis for the Hooke and Jeeves direct search method a C code obtained from netlib.org, which was just tailored to the case of interest. The code does not exactly follow the Hooke and Jeeves method as described in the preceding section 5.3.6.2, but implements a close version of the basic algorithm, with some enhancements.

At the α version stage, a single (monolithic) 'wrapper' routine was developed to make use of the aforementioned code to perform the following tasks:

- interface with MatCalc's codebase to set up the simulation environment (that is, using the codebase directly as a library, and *not* using MatCalc CLI as a tool)
- set up the 'cost' function for minimization
- launch the direct search method

The resulting call graph for this routine is depicted in Fig. 25 (generated automatically by Doxygen):

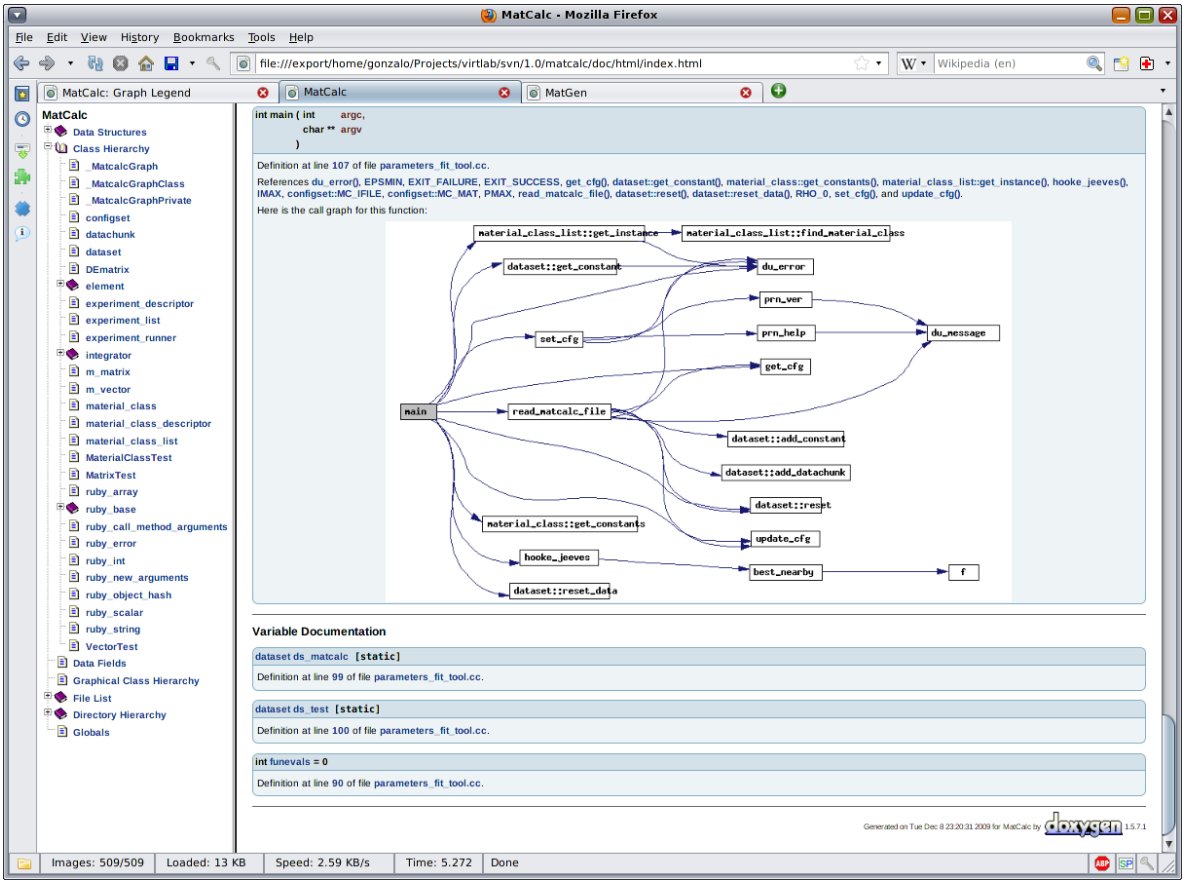


Fig. 25: Call graph for the Parameter Fitting Tool for MatCalc Material Models, generated by Doxygen

From the call graph, it can be seen that several classes belonging to MatCalc's codebase are being used to set and retrieve information in the data structures that the simulation engine uses to configure the current session and store the simulation results, get and set the material constants that are the parameters to be fitted by the Hooke and Jeeves algorithm, and access the simulation data files to set up the initial run.

The decision to use the codebase as a library in lieu of using the already developed MatCalc CLI frontend as a tool, was that of simplicity and performance: using MatCalc's CLI frontend would have implied that all the parameter updates – including those to update the trial centres and basepoints – performed file read/write and parse operations, whereas parameter updating withing a data structure in memory is straightforward. The underlying organization of the library, in fact, encouraged and greatly facilitated the latter approach.

[5.3.6.4] Tool preliminary performance and results

> **Note**: to the date of writing of the present report, the parameter fitting tool has been tested only superficially, since its usage unveiled a subtle but severe memory leak problem in the underlying library, which therefore had to be solved in the first place (see p. 128 in section 6.4 for a description of the issue, and section 9.13, p. 194 for the corresponding diagnose and fix).

For the initial testing of the tool's operation, a test case was 'forged' in Octave. Based on a 'closed form' solution for a specific material model behaviour, the resulting data was contaminated by random noise with the intention to emulate real world 'experimental' data (Fig. 26):

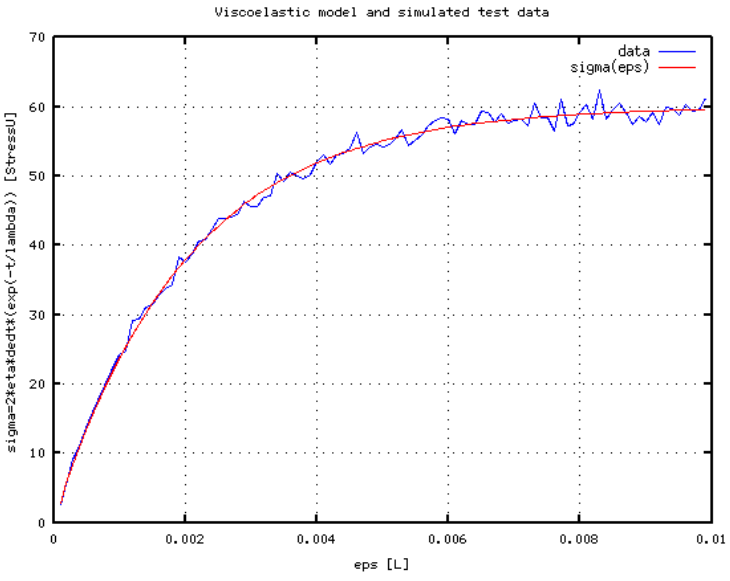


Fig. 26: Viscoelastic material 'experimental' test data (blue jagged trace) produced with Octave by contamination of a closed form solution (red smooth trace) with random noise.

The ‘true solution’ (the samples corresponding to the red smooth trace in Fig. 26) was produced with the following values for the two parameters of interest in the material model:

- `ElasticE = 3.0E+04`
- `eta = 3.0E+03`

The ‘real data’ produced by contamination of the ‘true solution’ with noise (the samples corresponding to the blue jagged trace in Fig. 26) was copied into the appropriate columns of a valid MatCalc file (like the one depicted in Fig. 18, p. 70). This file was then read in by the tool, and the data therein taken as the ‘reference’ data against which to perform the fitting.

A second MatCalc file, with its header set up for the adequate material model and virtual material test, was provided to the fitting tool, where the initial guess for the parameters was the following:

- `ElasticE_0 = 1.3E+03`
- `eta_0 = 5.5E+04`

Fig. 27 shows the tool’s output upon reading the simulation file, with the initial guesses indicated[48]:

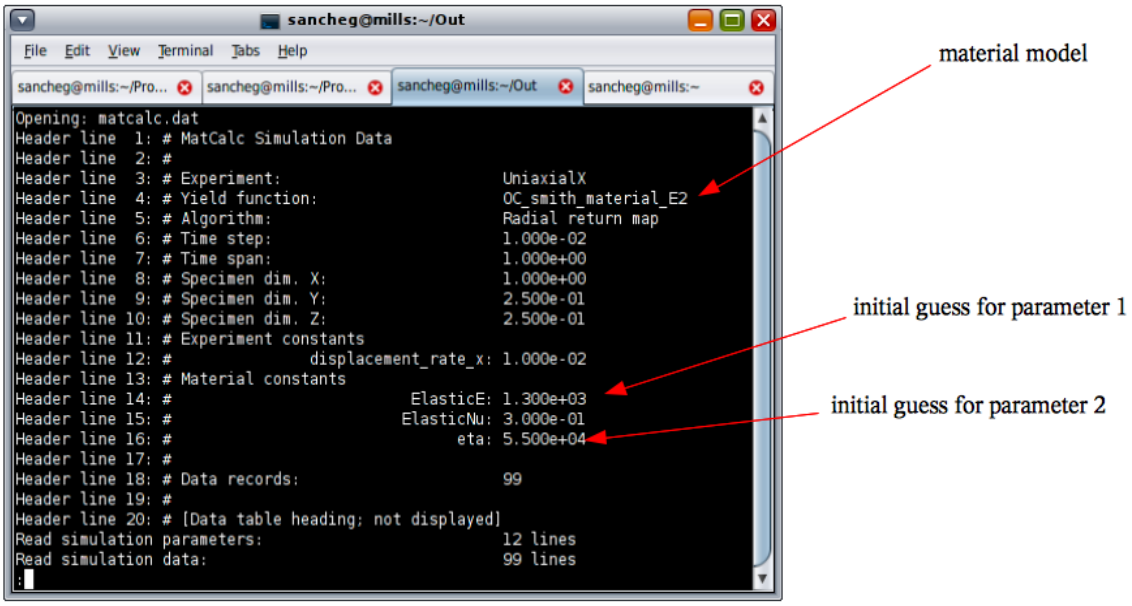

sancheg@mills:~/Out
File Edit View Terminal Tabs Help
sancheg@mills:~/Pro...
sancheg@mills:~/Pro...
sancheg@mills:~/Out
sancheg@mills:~
Opening: matcalc.dat
Header line 1: # MatCalc Simulation Data
Header line 2: #
Header line 3: # Experiment: UniaxialX
Header line 4: # Yield function: OC_smith_material_E2
Header line 5: # Algorithm: Radial return map
Header line 6: # Time step: 1.000e-02
Header line 7: # Time span: 1.000e+00
Header line 8: # Specimen dim. X: 1.000e+00
Header line 9: # Specimen dim. Y: 2.500e-01
Header line 10: # Specimen dim. Z: 2.500e-01
Header line 11: # Experiment constants
Header line 12: # displacement_rate_x: 1.000e-02
Header line 13: # Material constants
Header line 14: # ElasticE: 1.300e+03
Header line 15: # ElasticNu: 3.000e-01
Header line 16: # eta: 5.500e+04
Header line 17: #
Header line 18: # Data records: 99
Header line 19: #
Header line 20: # [Data table heading; not displayed]
Read simulation parameters: 12 lines
Read simulation data: 99 lines
material model
initial guess for parameter 1
initial guess for parameter 2

Fig. 27: The parameter fitting tool’s console output upon reading a MatCalc file containing the simulation setup and initial guesses.

As the search progresses, the FEM algorithm output is interspersed with that of the fitting tool[49] (Fig. 28).

[48] The third parameter, `ElasticNu`, is *not* of interest for the fitting objective, so regardless that the algorithm can handle that parameter, it was purposefully ignored in the tool’s implementation. Also note that even though the file contains data, it is ignored by the tool.

[49] This output can be filtered with a simple `grep` command to eliminate the FEM algorithm information, if desired.

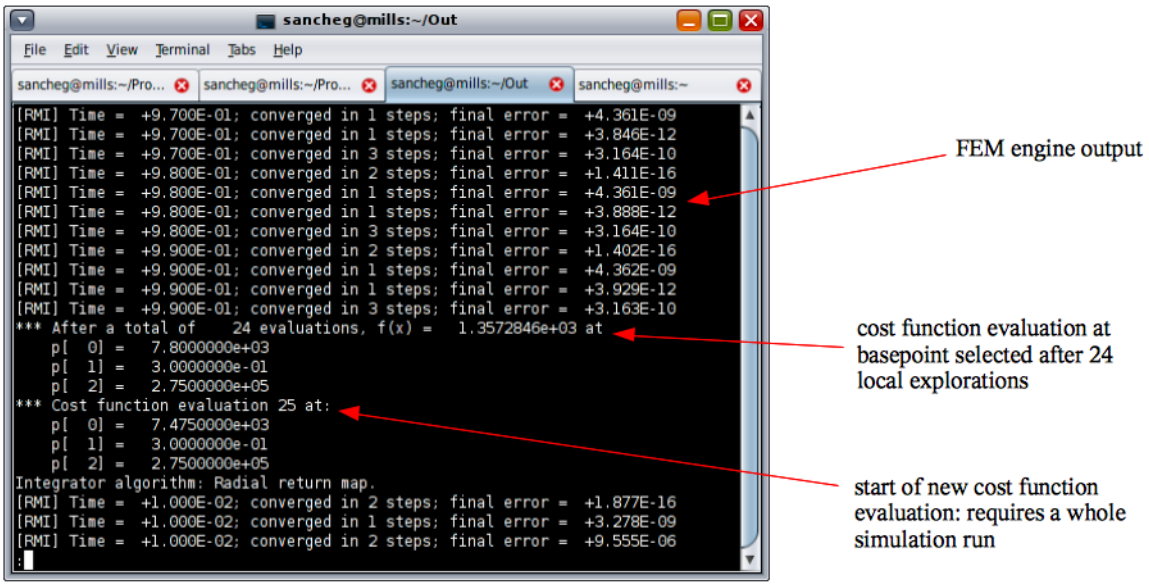

sancheg@mills:~/Out
File Edit View Terminal Tabs Help
sancheg@mills:~/Pro...
sancheg@mills:~/Pro...
sancheg@mills:~/Out
sancheg@mills:~
[RMI] Time = +9.700E-01; converged in 1 steps; final error = +4.361E-09
[RMI] Time = +9.700E-01; converged in 1 steps; final error = +3.846E-12
[RMI] Time = +9.700E-01; converged in 3 steps; final error = +3.164E-10
[RMI] Time = +9.800E-01; converged in 2 steps; final error = +1.411E-16
[RMI] Time = +9.800E-01; converged in 1 steps; final error = +4.361E-09
[RMI] Time = +9.800E-01; converged in 1 steps; final error = +3.888E-12
[RMI] Time = +9.800E-01; converged in 3 steps; final error = +3.164E-10
[RMI] Time = +9.900E-01; converged in 2 steps; final error = +1.402E-16
[RMI] Time = +9.900E-01; converged in 1 steps; final error = +4.362E-09
[RMI] Time = +9.900E-01; converged in 1 steps; final error = +3.929E-12
[RMI] Time = +9.900E-01; converged in 3 steps; final error = +3.163E-10
*** After a total of 24 evaluations, f(x) = 1.3572846e+03 at
p[0] = 7.8000000e+03
p[1] = 3.0000000e-01
p[2] = 2.7500000e+05
*** Cost function evaluation 25 at:
p[0] = 7.4750000e+03
p[1] = 3.0000000e-01
p[2] = 2.7500000e+05
Integrator algorithm: Radial return map.
[RMI] Time = +1.000E-02; converged in 2 steps; final error = +1.877E-16
[RMI] Time = +1.000E-02; converged in 1 steps; final error = +3.278E-09
[RMI] Time = +1.000E-02; converged in 2 steps; final error = +9.555E-06
FEM engine output
cost function evaluation at basepoint selected after 24 local explorations
start of new cost function evaluation: requires a whole simulation run

Fig. 28: Hooke & Jeeves search progress sample.

A summary of the search progress is saved in a log file, which for the test case being described was:

```
# Hooke & Jeeves search log
#
#  iters  funevals              p[0]              p[1]              p[2]
       1         1    1.30000000e+03    3.00000000e-01    5.50000000e+04
       2        24    7.80000000e+03    3.00000000e-01    2.75000000e+05
       3        61    9.10000000e+03    3.00000000e-01    1.37500000e+04
       4        99    1.17000000e+04    3.00000000e-01    6.87500000e+03
       5       116    1.19437500e+04    3.00000000e-01    6.87500000e+03
       6       128    1.19437500e+04    3.00000000e-01    5.15625000e+03
       7       309    2.38875000e+04    3.00000000e-01    3.43750000e+03
       8       320    2.38773438e+04    3.00000000e-01    3.00781250e+03
       9       592    3.06921875e+04    3.00000000e-01    3.00781250e+03
      10       643    3.05779297e+04    3.00000000e-01    3.00781250e+03
      11       665    3.05703125e+04    3.00000000e-01    3.00781250e+03
      12       671    3.05703125e+04    3.00000000e-01    3.00781250e+03
      13       683    3.05706299e+04    3.00000000e-01    3.00781250e+03
      13       694    3.05706299e+04    3.00000000e-01    3.01452637e+03
#
# IMAX:      100
# FMAX:      900
# RHO_0:     5.00000000e-01
# EPSMIN:    1.00000000e-04
```

The log shows that the method succeeded in converging to the following point in the parameter space[50], which differs from the 'true' solutions in about 2% and 0.5%, respectively:

- `ElasticE_1 = 3.05706299e+04`
- `eta_1 = 3.01452637e+03`

[50] The 'coordinates' of the point corresponding to the `ElasticNu` parameter (`p[1]`) are ignored, since the algorithm was purposefully 'hacked' to ignore that specific parameter (see note 48).

For the test case selected, which is two dimensional, the 'search evolution' can be plotted from the third and fifth columns of the above search log file (Fig. 29):

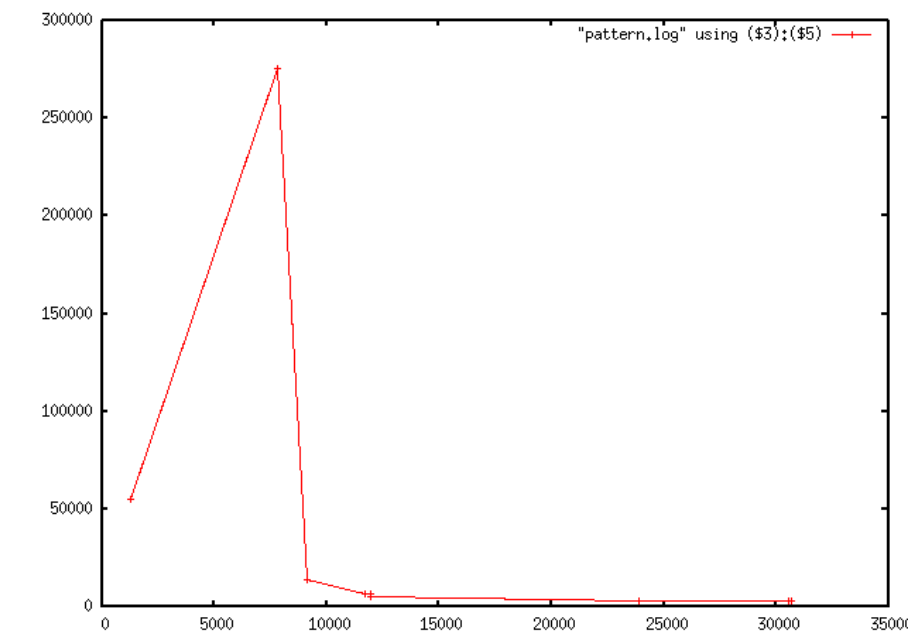


Fig. 29: Search progress plot (X axis: `ElasticE`; Y axis: `eta`)

The plot in Fig. 29 cannot provide much detail, given that the algorithm implementation used does not return information on the trial centres tested, as the sample search path in Fig. 23 (p. 83) shows. An implementation of the algorithm following the description provided in [SHL83] (see section 5.3.6.2) was developed, with the result that a finer insight into the search is provided.

The insight comes at a cost: the 'custom' algorithm is far more inefficient than the algorithm originally downloaded from netlib.org, as the following excerpt from the resulting log shows[51]:

```
# Hooke & Jeeves pattern search log #
# eps:   1.00000000e-03
# rho:   5.00000000e-01
# r_max: 1000
#
# r  f_ev            f(p)             p[0]             p[1]             p[2]
  0     1   1.86084132e+05   1.30000000e+03   3.00000000e-01   5.50000000e+04
  1     5   1.57549755e+05   1.95000000e+03   3.00000000e-01   8.25000000e+04
  2     9   1.08670337e+05   3.25000000e+03   3.00000000e-01   1.37500000e+05
  .
  .     (intermediate output omitted here)
  .
174   915   1.09503500e+02   3.08934082e+04   3.00000000e-01   3.00781250e+03
175   921   1.09503500e+02   3.08934082e+04   3.00000000e-01   3.00781250e+03
176   927   1.09503500e+02   3.08934082e+04   3.00000000e-01   3.00781250e+03
176   927   1.09503500e+02   3.08934082e+04   3.00000000e-01   3.00781250e+03
# EOF
```

[51] Note the file layout has been modified in the last tried version of the algorithm.

Fig. 30 clearly shows the 'step size reduction', which gives an idea of the time progress of the search – painfully slow once `eta` has reached an almost stable value circa `3E+3` and `ElasticE` is about `10E+3`:

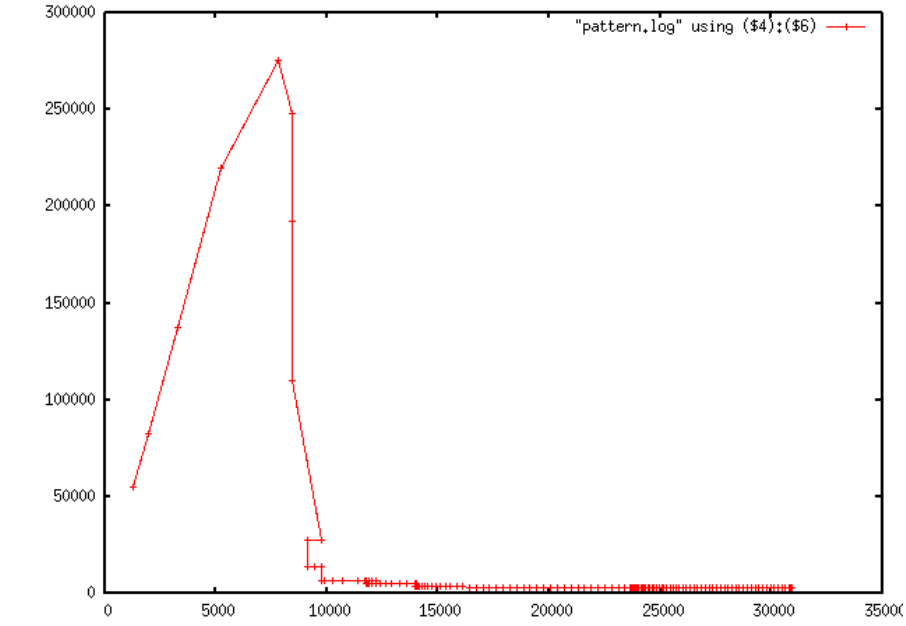


Fig. 30: Search progress plot (X axis: `ElasticE`; Y axis: `eta`) with custom algorithm.

In terms of the number of cost function evaluations, the original algorithm required 694 versus the 927 required by the 'custom' version. Already the original algorithm takes 20 minutes of wall-clock time to run on CAS server `mills.mcmaster.ca`, and the 'custom' one surpasses the half hour for the test case used.

Additionally, some specific choices for the original algorithm convergence/stepsize factors can produce completely wrong results for the same initial guesses, as the following log shows:

```
# Hooke & Jeeves search log
#
#  iters  funevals             p[0]             p[1]             p[2]
       1         1    1.30000000e+03    3.00000000e-01    5.50000000e+04
       2        48    8.45000000e+03    3.00000000e-01    3.46500000e+05
       3        71    8.34600000e+03    3.00000000e-01    3.41000000e+05
       4        98    8.36550000e+03    3.00000000e-01    3.40175000e+05
       5       113    8.36511000e+03    3.00000000e-01    3.40158500e+05
       6       136    8.36524000e+03    3.00000000e-01    3.40153000e+05
       7       155    8.36523220e+03    3.00000000e-01    3.40152670e+05
       8       186    8.36523493e+03    3.00000000e-01    3.40152554e+05
       8       213    8.36523473e+03    3.00000000e-01    3.40152546e+05
#
# IMAX:      100
# FMAX:      1000
# RHO_0:     1.00000000e-01
# EPSMIN:    1.00000000e-08
```

It is clear in the light of the above results that there is significant work to be done with regards to this tool; this is suggested in the following section 5.3.6.5.

[5.3.6.5] Future work

Currently, the tool can handle material models with arbitrary number of parameters, but can only fit against a *single* real world material test run; therefore, the fit is done as per the cost function described in Eq. 8. Future work should address *multiple* material tests, so that the fitting is done as per the cost function in Eq. 9 instead.

There is a clear need for performance improvement in the material model parameters fitting tool, as well as a need to investigate possible non-convergence issues, or convergence to wrong results. At the time of writing of this report, there is a project encompassed within the CES#713 2010 course at CES Department (McMaster University) "*The Message Passing Interface for Parallel Applications*", that will target the parallelization of the search algorithm by application of the Message Passing Interface (MPI) constructs for building parallel applications [San10]. The parallelization of the search algorithm is expected to improve the following:

- **Performance**: By the mere evaluation of the cost function in parallel for each trial centre tested by the algorithm, a significant time saving is expected if there are enough cores or nodes in the system for each possible coordinate 'shift'[52]. Also, provided enough computing power is available in the system, several step sizes for each coordinate 'shift' can be tried in parallel with the former parallelization scheme. Furthermore, for when the tool is able to address several real world material tests, all of the work in each case could be executed in parallel, with appropriate communication of the minimization search between all processes.
- **Convergence**: Parallel execution of the tool for different initial guesses can be used to check if the convergence is towards the same local minimum, or different local minimums. The challenge in this case is the termination criteria for the overall search, in the case of quicker convergence of one path with respect to the rest, compounded with the possibility of hitting different local minima.

A complementary approach to improving the performance would be to study the possibility of progressively augmenting the step size once a pattern has been found, as is the case shown in Fig. 29 (p. 88) and Fig. 30 (p. 89) after `eta` has reached an almost stable value around `3E+3` and `ElasticE` is about `10E+3`. Care should be taken, though, to avoid possible 'oscillations' in the resulting algorithm step-size control, that could lead to non-convergence.

[52] For the two parameter case, a four core machine could potentially perform all computations in parallel (MPI code can work on SMP machines as well as clusters, or combinations thereof).

[5.3.7] Documentation

> "*Industrial documentation suffers from problems so serious that programmers rely almost exclusively on the code. To remedy this problems, the documents must be designed as carefully as the code they support. The same set of documents must be used for design, implementation, and maintenance.*
>
> *Our design methodology is based on the central role of documentation. It is, in some ways, more important than the code. Discard the code and keep the documents, and you can recreate the code quickly and capably. Discard the documents and keep the code, and the resulting system will be hard to control and expensive to maintain.*" ([HS99], p. 16)

To address the maintainability, understandability and reusability qualities of Virtlab library − and, as a consequence, its verifiability and reliability − specific effort was dedicated to improve the software package documentation.

As of May 2009, the documentation for Virtlab library was limited to the original developer's M.Sc. Thesis [McC07] and a few text files that addressed some very specific issues of the software library. In the period from May 2009 to March 2010, the following documentation elements were created and/or improved:

- Doxygen generated documentation for the codebase.
- `INSTALL2`, `TODO`, and `ChangeLog` files.
- Subversion commit log.
- Other text files for specific issues.

These documentation items are described in the following sections.

[5.3.7.1] Doxygen codebase documentation

The documentation generated by Doxygen [vHe97] is an organized collection of HTML files that are automatically generated by parsing the software's source code files

An overview of Doxygen is provided in section 6.3, p. 115.

The generated documentation set includes:

- complete source code files listing.
- index of data structures (class hierarchy and data fields).
- folder/directory hierarchy.
- indexes of all functions, structures, unions, enumerations, variables, type definitions, and definitions.

Graphics are provided for all class inheritance diagrams, dependency graphs, and function call/caller graphs. The generated documentation pages use syntax highlighting and/or colour coding for enhanced readability, and all objects therein are cross-referenced through hyper text links to facilitate browsing. Due to its hyper-linked structure and extension, it is pointless and impractical to include the generated documentation as an appendix to the

present report. Fig. 25 (p. 84); Fig. 35 (p. 118); Fig. 36 (p. 119); Fig. 37 (p. 120) and Fig. 38 (p. 121) show screenshots of a typical Doxygen generated pages.

> For convenience, the documentation corresponding to the latest snapshot of the code can be viewed by pointing a web-browser to the following locations:
>
> http://www.cas.mcmaster.ca/~sancheg/MatCalc-1.1
>
> http://www.cas.mcmaster.ca/~sancheg/MatGen-1.1

The generation options for Doxygen are defined in a configuration file, which for the case of MatCalc and MatGen has been tailored appropriately. Most of the source code files in Virtlab's library have been provided with a brief comment about the file's purpose, following Doxygen's expected format. This specially formatted comments are automatically parsed by Doxygen and used for the generated HTML documentation. Several of the source code files have also been provided with an extended comment in Doxygen's expected format, so that are also included appropriately in the generated HTML documentation. The comments included in the source code files try to warn about the state of knowledge of the file, given that most of the original codebase was undocumented. The table in section 9.12 (p. 191) – copied from that generated by Doxygen – lists the files that comprise MatCalc, alongside the brief comments for each of the (documented) files, where the said warnings can be observed.

To further facilitate the documentation maintenance, two very simple shell scripts (not documented in this report) have been created to automate the corresponding documentation update, and are distributed with the software package.

Doxygen by itself can generate plain HTML documentation, but to produce certain graphical output (such as the graphs shown in Fig. 25, Fig. 35, Fig. 36 and Fig. 37) it requires other tools that are not usually part of a typical Linux or OpenSolaris distribution, nor are by default part of MacOS or Solaris environments. It was therefore decided that a `.tar` 'Unix tape archive' file, containing the generated documentation for each of MatCalc and MatGen, is to be included with Virtlab software package, as a convenience for a developer who might not have the complete set of tools to generate the full-fledged Doxygen documentation the first time he or she accesses the codebase.

[5.3.7.2] `ChangeLog` and `TODO` files

Not altogether following the GNU coding standards, but taking the ideas therein, an attempt to maintain proper `ChangeLog` and `TODO` files was effected.

The `ChangeLog` files are intended as a log of all changes done to the codebase, reported chronologically and on a file per file basis, alongside the developer that introduced the change.

> Sections 9.1 and 9.3 contain a copy of the `ChangeLog` files for MatCalc and MatGen, respectively, to the date of writing of the present report.

Arguably, the purpose and contents of these files duplicate those of the Subversion commit log (see section 5.3.7.4) and as such, can lead to maintenance problems and – what is worse – inconsistencies between the said log files. For the particular case of Virtlab, the `ChangeLog` files were maintained for the sake of keeping MatCalc and MatGen issues in separate files, given the fact that the repository for the code – and hence, its corresponding commit log – is unique.

The `TODO` files are intended to keep a list of tasks to be done on the software's codebase. This range from needed bug fixes, to enhancements, or ideas for experimenting, and are meant as a reminder to the developers that might be assigned work on the codebase. Sections 9.2 and 9.4 contain a copy of the `TODO` files for MatCalc and MatGen, respectively, to the date of writing of the present report.

[5.3.7.3] `INSTALL2` file

As mentioned in section 5.3.1, towards the portability quality, a file `INSTALL2` with 'advanced' building and installation instructions was generated and added to Virtlab software package. In particular, compile time and run time linking issues are described for the different platforms.

> Section 9.5 in the Appendix lists the complete contents of the file.

[5.3.7.4] Subversion commit log

Virtlab codebase is managed under the Subversion (SVN) version control system [Col00].

> An overview of version control systems is provided in section 6.1, p. 102.

As other contemporary version control tools, Subversion forces each update of the repository to be accompanied by a corresponding 'commit message'. The developer or developers are supposed to use this message instance to describe the changes effected to the files being updated in the codebase repository. Thus, the log file that is generated can act as a `ChangeLog` file for the project under version control. The clear advantage of this log over a usual `ChangeLog` file is that the developer is forced to write a message whenever he or she pretends to alter the software's codebase in the repository, his or her name is associated automatically with the change, and the tool already writes into the message file the list of files that have been modified locally with respect to those in the software's repository (which helps the developer to remember what were the actual files that changed he/she did since the last commit operation).

Given that during the period from May 2009 to March 2010 there was only one developer working on the software codebase, and that specific `ChangeLog` files were maintained, the Subversion logs are far from being a quality document.

It is also worth noting that – ironically, against the basic purpose of a version control system – the Subversion repository for VirtLab has been divided 'by hand' into 4 folders, which contain the repository state up to some 'milestones' in the development process. The rationale for the 'hand made' aforementioned division is that those four major stages in the

development of MatCalc/MatGen would stand out immediately and clearly, avoiding the need to execute SVN commands to surf the SVN logs and checkout different revisions. Even though this could have easily been accomplished by a proper use of the SVN resources (tags, branches, comments) those resources were not properly exploited from the beginning, and the task of rewinding the revisions to fix them was not worth undertaking on the face of other prioritary documentation and coding issues.

[5.3.7.5] Other support text files

Some issues pertaining to Virtlab internals, originally documented in specific documentation text files, were leveraged by adding information from the main document existent as of May 2009 [McC07].

These files are about:

- The way to use Virtlab modules as a library for material simulation.
- How to add a material test to MatCalc.
- How to add a material model to MatCalc.
- Generating material models with MatGen.

> Sections 9.6, 9.7, 9.8, and 9.9 list the contents of the aforementioned files.

[5.3.7.6] Future work

Future work in documentation should address the following issues:

- Complete the source code documentation 'brief' and 'detailed' comments, for use by the Doxygen documentation tool.
- Document the objects (classes, methods, functions, structures, etc.) in the codebase using Doxygen's appropriate keywords, to improve the automatically generated documentation thereafter.
- Automate the maintenance of the documentation set. One step towards this goal would be to study the possibility of using Doxygen to automatically extract '`TODO`' and '`FIXME`' comments within the code, with the purpose of automating the maintenance of the `TODO` tasks.
- Produce the complete set of documents listed in Table 1 (p. 15). The **VL_TM** document should actually be a *document set* − **System Requirements Specification (SRS)**; **Module Guide (MG)**: **Module Interface Specification (MIS)** − as suggested in [HS99]. Any step towards automating the maintenance of this document set should be pursued.

[5.3.8] Virtlab tests and tools

Virtlab software library included a number of files that, even undocumented, clearly were intended either as tests for specific aspects of the codebase, or as tools for accessing specific information of the software library and checking simulation results.

Various of these tests and tools have been refactored to adapt to the changes in the codebase − particularly those effected towards improving the separation of concerns (section 5.3.5, p. 76) − some others have been improved, and some have been completely rewritten or developed from scratch. The following sections highlight the major tests and tools changes.

[5.3.8.1] `lists` tool

The `lists` tool, which informs the user of the material tests and material models available for simulation (alongside with their particular constants) has been improved from its original version for providing a better console output, and has been refactored to avoid memory leaks (although some potential problems still remain; see p. 126 in section 6.4):

```
gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src$ ./lists

--- Supported Material Models ------------------------

[1]   Material:   OC_dp_material
      Constants:
            (1)   ElasticE
            (2)   ElasticNu
            (3)   c
            (4)   phi
            (5)   psi

[2]   Material:   OC_hardening_material
      Constants:
            (1)   ElasticE
            (2)   ElasticNu
            (3)   A
            (4)   m
            (5)   n

[3]   Material:   OC_plfluid_material_E3
      Constants:
            (1)   ElasticE
            (2)   ElasticNu
            (3)   A
            (4)   m

[4]   Material:   OC_smith_material_E2
      Constants:
            (1)   ElasticE
            (2)   ElasticNu
            (3)   eta

[5]   Material:   OC_vonmises_material_E5
      Constants:
            (1)   ElasticE
            (2)   ElasticNu
            (3)   Kyield


--- Available Experiments ----------------------------

Booted Ruby.
[1]   Experiment: UniaxialX
```

```
      Constants:
            (1)   displacement_rate_x

[2]   Experiment: UniaxialY
      Constants:
            (1)   displacement_rate_y

[3]   Experiment: UniaxialXRC
      Constants:
            (1)   displacement_rate_x
            (2)   displacement_rate_x_initial_v
            (3)   displacement_rate_x_max_v
            (4)   displacement_rate_x_min_v
            (5)   displacement_rate_x_dv

[4]   Experiment: UniaxialZ
      Constants:
            (1)   displacement_rate_z

[5]   Experiment: Shear
      Constants:
            (1)   displacement_rate_shear

[6]   Experiment: Triaxial
      Constants:
            (1)   lx
            (2)   ly
            (3)   lz
            (4)   sigma_radial
            (5)   sigma_axial
            (6)   alpha

[7]   Experiment: UniaxialXForce
      Constants:
            (1)   uniaxial_force

[8]   Experiment: UniaxialXSO
      Constants:
            (1)   displacement_rate_x
            (2)   displacement_rate_x_shut_off_time
            (3)   displacement_rate_x_turn_on_time
```

[5.3.8.2] `dset_diff` tool

The `dset_diff` tool (originally, `difftool`) which compares two datasets, has been refactored to use the new file input module[53], and has been extended to compare shear stresses/strains:

```
gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src$ ./dset_diff
Usage: dset_diff <datafile1.dat> <datafile2.dat>

gonzalo@sancheg:~/Projects/virtlab/svn/1.1/matcalc/src$ ./dset_diff test.dat output.dat
Opening: test.dat
Header line  1: # MatCalc Simulation Data
Header line  2: #
Header line  3: # Experiment:                        UniaxialX
Header line  4: # Yield function:                   OC_smith_material_E2
Header line  5: # Algorithm:                        Radial return map
Header line  6: # Time step:                        1.000e-02
Header line  7: # Time span:                        1.000e+00
```

[53] This is apparent from the console output, which is identical to that of MatCalc's GUI ad CLI frontends upon reading an input file (*c.f.* Fig. 15, p. 66).

```
Header line  8: # Specimen dim. X:                   1.000e+00
Header line  9: # Specimen dim. Y:                   2.500e-01
Header line 10: # Specimen dim. Z:                   2.500e-01
Header line 11: # Experiment constants
Header line 12: #               displacement_rate_x: 1.000e-02
Header line 13: # Material constants
Header line 14: #                          ElasticE: 3.000e+04
Header line 15: #                         ElasticNu: 3.000e-01
Header line 16: #                               eta: 3.000e+03
Header line 17: #
Header line 18: # Data records:                      99
Header line 19: #
Header line 20: # [Data table heading; not displayed]
Read simulation parameters:                         12 lines
Read simulation data:                               99 lines
Opening: output.dat
Header line  1: # MatCalc Simulation Data
Header line  2: #
Header line  3: # Experiment:                       UniaxialX
Header line  4: # Yield function:                   OC_smith_material_E2
Header line  5: # Algorithm:                        Radial return map
Header line  6: # Time step:                       1.000e-02
Header line  7: # Time span:                       1.000e+00
Header line  8: # Specimen dim. X:                  1.000e+00
Header line  9: # Specimen dim. Y:                  1.000e+00
Header line 10: # Specimen dim. Z:                  1.000e+00
Header line 11: # Experiment constants
Header line 12: #               displacement_rate_x: 1.000e-02
Header line 13: # Material constants
Header line 14: #                          ElasticE: 3.000e+04
Header line 15: #                         ElasticNu: 3.000e-01
Header line 16: #                               eta: 3.000e+03
Header line 17: #
Header line 18: # Data records:                      99
Header line 19: #
Header line 20: # [Data table heading; not displayed]
Read simulation parameters:                         12 lines
Read simulation data:                               99 lines

*** Comparing data in 'test.dat' (reference) with data in 'output.dat':
    relative error stress  X:   9.757E-03
    relative error strain  X:   0.000E+00

    relative error stress  Y:   0.000E+00
    relative error strain  Y:   0.000E+00

    relative error stress  Z:   0.000E+00
    relative error strain  Z:   0.000E+00

    relative error stress XY:   8.539E-01
    relative error strain XY:   3.990E-01

    relative error stress XZ:   8.539E-01
    relative error strain XZ:   3.990E-01

    relative error stress YZ:   4.236E-10
    relative error strain YZ:   0.000E+00
```

[5.3.8.3] `configset_test` utility

A `configset_test` test was created to check the workings of the newly created `configset` module. It essentially exercises the allocation and deallocation of the members of type `string` in the configuration structure, and the use of the 'set' and 'get' functions for each of the structure members.

[5.3.8.4] `FEM_test` tool

The `FEM_test` tool (originally, `femtest`) has been completely rewritten, and is now a 'standalone' CLI tool that performs the same FEM simulation as MatCalc's CLI frontend. Besides accepting an input file for defining the simulation, `FEM_test` by default computes a simulation for a linear viscoelastic material model, under an Uniaxial test; for that particular setup, it admits a number of command line switches to 'tweak' the simulation that the normal CLI frontend to MatCalc does not accept[54], as can be seen in Fig. 31:

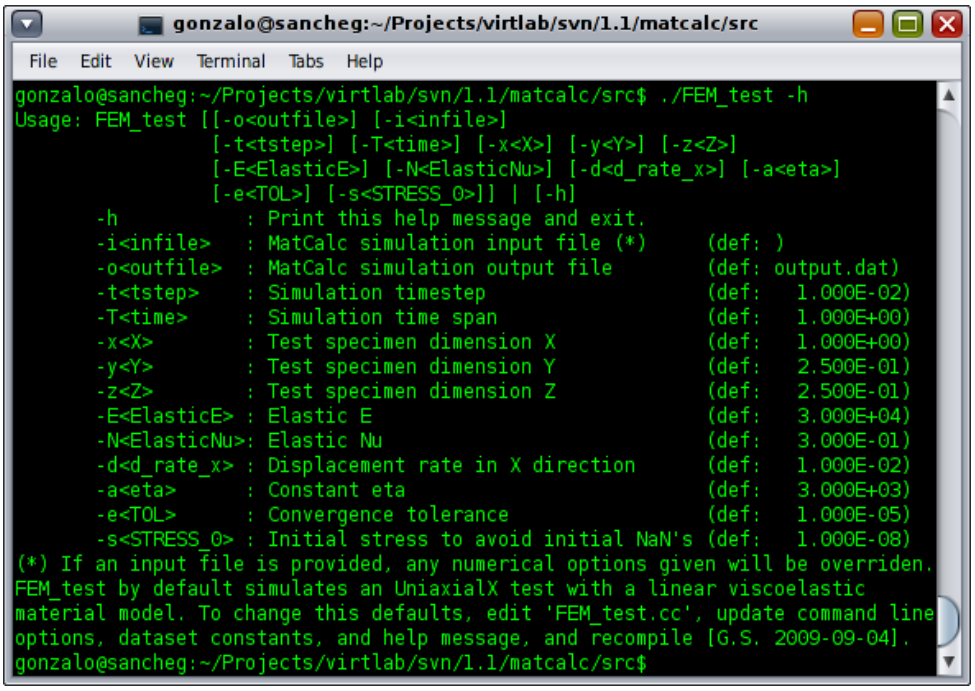


Fig. 31: `FEM_test` help message.

The linear viscoelastic model 'hardcoded' into `FEM_test` was selected because it has a 'closed form' (analytical) solution against which the simulation results can be compared.

This tool, alongside with the Material Model Parameters Fitting Tool described in section 5.3.6 (p. 81) and the `dset_diff` tool described above, is geared towards the verifiability and reliability software qualities of Virtlab library.

> **Note**: `FEM_test` does not use the `input`, `output`, or `configset` modules developed for the library. This is product of a specific design decision, to make it independent as much as possible of the underlying library modules except those *directly* involved with the simulation – that is, the FEM engine – since the purpose of the tool is to test the simulation engine itself.

[54] It also permits changing two parameters used by the integration algorithm, `TOL` and `STRESS_0`, which are not accessible by neither of MatCalc's GUI or CLI frontends.

> Yet, this decision comes along with a maintenance cost: *any changes to the output module that affects the file format has to be 'recoded' into* `FEM_test` *to maintain file format compatibility.*

[5.3.8.5] Future work

It has to be noted that the purpose and output of various of the original tests within the codebase are *not* immediately understandable, due to the lack of documentation[55]. The Doxygen documentation for these tests acknowledges the knowledge state for the test/tool to the moment of writing of the present report. The brief comment for each of the files can be checked to the end of the table posted in section 9.12, starting at p. 191. Naturally, among the future work with respect to the tools and tests for Virtlab software package is that of correctly comprehending and documenting the existent tests.

These tests, as any other tests that might be created, should be encompassed under adequate *unit testing* schemes, to better address the verifiability and reliability qualities of the software package. Currently, Virtlab library has only two modules (`Math_test` and `MC_test`) under a unit testing framework, CppUnit[56].

> "...***unit testing*** *is a software verification and validation method in which a programmer tests if individual units of source code are fit for use. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual function or procedure. Ideally, each test case is independent from the others.*"
>
> http://en.wikipedia.org/wiki/Unit_testing

Additionally, a proper *regression testing* suite should be created to the same purpose of systematically addressing the verifiability and reliability qualities of the software package.

> ***Regression testing*** *is any type of software testing that seeks to uncover software errors by partially retesting a modified program. The intent of regression testing is to provide a general assurance that no additional errors were introduced in the process of fixing other problems. Regression testing is commonly used to efficiently test the system by systematically selecting the appropriate minimum suite of tests needed to adequately cover the affected change. Common methods of regression testing include rerunning previously run tests and checking whether previously fixed faults have re-emerged. "One of the main reasons for regression testing is that it's often extremely difficult for a programmer to figure out how a change in one part of the software will echo in other parts of the software."[1]* [57]
>
> http://en.wikipedia.org/wiki/Regression_testing

[55] And to the fact that not enough time during the period May 2009 - March 2010 was allocated to review the code, due to other priorities for the work during the said period.

[56] http://sourceforge.net/apps/mediawiki/cppunit/index.php?title=Main_Page

[57] **[1]** Savenkov, Roman (2008). *How to Become a Software Tester*. Roman Savenkov Consulting. p. 386. ISBN 978-0-615-23372-7.

Note: An *ad-hoc* attempt at regression testing was started with the creation of a shell script, `test_matcalc.sh`, that runs a series of memory leak checks (using the memory checker tool in Valgrind [Sew00]) on various MatCalc v. 1.1 executables: `lists`, `dset_diff`, `configset_test`, `FEM_test`, `Ruby_test`, and `matcalc` itself (section 6.4, p. 123 shows some examples of the application of Valgrind to the referred tools).

For the memory leaks check, just depending on the different exit paths the executable has, at least a check has to be performed to verify if that particular termination path has not produced a leak.

The script (listed in section 9.10, p. 184) gives an idea of the various testing combinations the referred executables can undergo for this leak check.

[6] SOFTWARE TOOLS

This chapter presents an overview of key software tools that were used in the Virtlab Project. Most of the contents herein have been selected from the respective tools websites, manuals, or articles describing the tools usage. This reference information is intended for readers unfamiliar with the tools' purpose and/or basic usage. Examples of the use of these tools in Virtlab project are presented whenever appropriate.

[6.1] Subversion

Subversion is a version control system, that can be used to manage any collection of computer files [Col00].

> "***Revision control, also known as version control, source control or (source) code management (SCM), is the management of changes to documents, programs, and other information stored as computer files***. *It is most commonly used in software development, where a team of people may change the same files.* ***Changes are usually identified by a number or letter code, termed the "revision number"****, "revision level", or simply "revision". (...) Each revision is associated with a timestamp and the person making the change.* ***Revisions can be compared, restored, and with some types of files, merged***."[58]
>
> http://en.wikipedia.org/wiki/Revision_control

Version control systems basically contain the *history* of each of the files under version control, so that at any time, any file can be 'rolled back' to a previous state in history.

> "*As teams design, develop and deploy software, it is extremely common for multiple versions of the same software to be deployed in different sites, and for the software's developers to be working simultaneously on updates. Bugs and other issues with software are often only present in certain versions (because of the fixing of some problems and the introduction of others as the program develops). Therefore, for the purposes of locating and fixing bugs, it is vitally important to be able to retrieve and run different versions of the software to determine in which version(s) the problem occurs. It may also be necessary to develop two versions of the software concurrently (for instance, where one version has bugs fixed, but no new features (branch), while the other version is where new features are worked on (trunk).*"
>
> "*At the simplest level, developers could simply retain multiple copies of the different versions of the program, and label them appropriately. This simple approach has been used on many large software projects. While this method can work, it is inefficient as many near-identical copies of the program have to be maintained. This requires a lot of self-discipline on the part of developers, and often leads to mistakes. Consequently, systems to automate some or all of the revision control process have been developed.*"
>
> http://en.wikipedia.org/wiki/Revision_control

There are basically two different approaches to version control systems: the traditional *centralized* model, and the *distributed* model:

- The *centralized* model relies on a server to hold the codebase 'repository', from where each developer copies the source code files to work upon.
- The *distributed* approach relies on the copies of the codebase distributed in every developer's workstation.

[58] The boldface type applied here if for 'highlighting' sections of particular relevance in the quote; it is *not* part of the original text.

Subversion is a version control system of the centralized type, requiring a server for the repository. It is the revision control system with which Virtlab library was originally managed.

From the developer's perspective, the initial step to start working with a codebase under version control in a centralized version control system like Subversion is the following:

- Copy the contents of the repository (located in the file server) into a working folder (the developer's local hard disk drive)

After 'acquiring' the initial working copy, the developer's subsequent workflow is basically a repetition of the following steps:

- Update the working copy with the latest state of the repository.
- Make changes to the code in the working folder.
- Update the repository to incorporate those changes.

The centralized type of version control system manages the possible overlapping operations of several developers on the repository, and keeps track of each change to every file in the repository, timestamped, and including the identity of the author of the changes.

> The use of a version control system to manage *all* files pertaining to a software package is perhaps the first tool that a developer should use, for addressing the maintenance of both *code* and *documentation.*

[6.2] Build Systems

"*The build tool is always just a piece in the larger software development system. Let us define more precisely the terms used below.* ***The build tool reads a build description and then directly starts other tools to actually produce the result of the build (a report, a compiled file, etc.)***. ***A build system is a set of tools, including a build tool*** *(by the way, a good build system will support several competing build tools). A build system may produce or adapt the build descriptions before they are used by the build tool. A build system may in some way manage the build result (for example, it may post the summary of an automatic build on a Web site).* ***Next to the build tool, the other important tool in a build system is the configuration tool, the one that adapts or automatically generates the build descriptions used by the build tool***.

To understand the paradigm shift introduced by build systems, it is important to first know the new requirements addressed by build systems: software distribution as a package of sources. *The typical usage scenario for a build tool is C sourcecode development. Here, the genuine input changes frequently and locally. Avoidance is a crucial feature in this use case. How building works elsewhere is less important. Also, the addition of new components to a build has to be easy. By contrast, for software distribution, the crucial need is the ability to customize the software. It has to be adapted to new build platforms, to new runtime platforms, to sitewide conventions, etc. The builds are far less frequent. (Avoidance is not really a requirement, for example.) The difference in requirements has had an influence on the evolutions of build tools (some being used mostly from within build systems, and some being used mostly stand-alone).*

The key advance in build systems is the fact that they put some executable code in the sourcecode distribution of the software to be built. We name the executable part in the source distribution the configuration tool. This changes the nature of the sourcecode distribution; now it is more like an installer package. *"Installer" is a term stolen from binary distributions. (For software distributed in binary form, it is a long-time established practice that the package has to "execute" on the system that receives the software distribution).* ***Because you are now supposed to execute some program delivered with the sources before you start the build, the source distribution has a chance to adapt itself to the current build machine. This is what makes build systems so helpful with sourcecode portability. The configuration tool may change build descriptions or sourcecode or both***. *Sometimes, the configuration tool has evolved into a complex interactive tool (see the several tools available to configure Linux kernel compilation).*

The configuration tool inspects the build platform in small steps, with individual checks for one feature. Hereafter, we will call these steps probes. *How results of probes are represented differs from one tool to another, but most of them have a caching mechanism in place so that you don't need to run a probe each time you need the result of that probe. Probes can have various granularities, can be independent or can be dependent on the result of other probes. Roughly speaking, the available set of probes is the way a configuration tool represents its knowledge about the platform to which it needs to adapt. The result of the entire set of probes defines de facto a model for a given*

platform. This hurts on the plane of operating systems, with more or less support from those systems' maintainers. ***Given the diversity of platforms to cover, configuration tools in general take on a tremendous job. This unavoidably generates frustration in different places and at different levels. Try to be positive and not judge a configuration tool only by its failures in some cases****."*[59]

http://freshmeat.net/articles/make-alternatives

[59] The boldface type used here is for 'highlighting' passages of the quote, but is *not* present in the original text.

[6.2.1] The GNU Build System

The GNU build system, also known as the Autotools, is a suite of programming tools produced by the GNU project. These tools are designed to assist in making various source-code packages portable to many Unix-like systems.

http://en.wikipedia.org/wiki/GNU_build_system

[6.2.1.1] Autoconf, Automake and Libtool

The GNU Build System is comprised mainly of the packages Autoconf, Automake and Libtool:

- ***Autoconf*** *produces a configuration shell script, named `configure', which probes the installer platform for portability related information which is required to customize makefiles, configuration header files, and other application specific files. Then it proceeds to generate customized versions of these files from generic templates. This way, the user will not need to customize these files manually.*
- ***Automake*** *produces makefile templates, 'Makefile.in' to be used by Autoconf, from a very high level specification stored in a file called `Makefile.am'. Automake produces makefiles that conform to the GNU makefile standards, taking away the extraordinary effort required to produce them by hand. Automake requires Autoconf in order to be used properly.*
- ***Libtool*** *makes it possible to compile position independent code and build shared libraries in a portable manner. It does not require either Autoconf, or Automake and can be used independently. Automake however supports libtool and interoperates with it in a seamless manner.*

([GJ03], p. 1)

[6.2.1.2] The `configure` shell script

The '**configure**' script produced by the Autoconf tool is the 'configuration tool' for the GNU build system that 'adapts or automatically generates the build descriptions used by the build tool' so that the software package, distributed as *source code*, behaves like an 'installer package' like the commonly used for distributing *executable* code (binary code).

In the case of the GNU Build system, `configure` is a *highly complex* shell script. Two sections of a sample `configure` script (that spans a total of 11761 lines) are shown on the screenshot in Fig. 32 (p. 107).

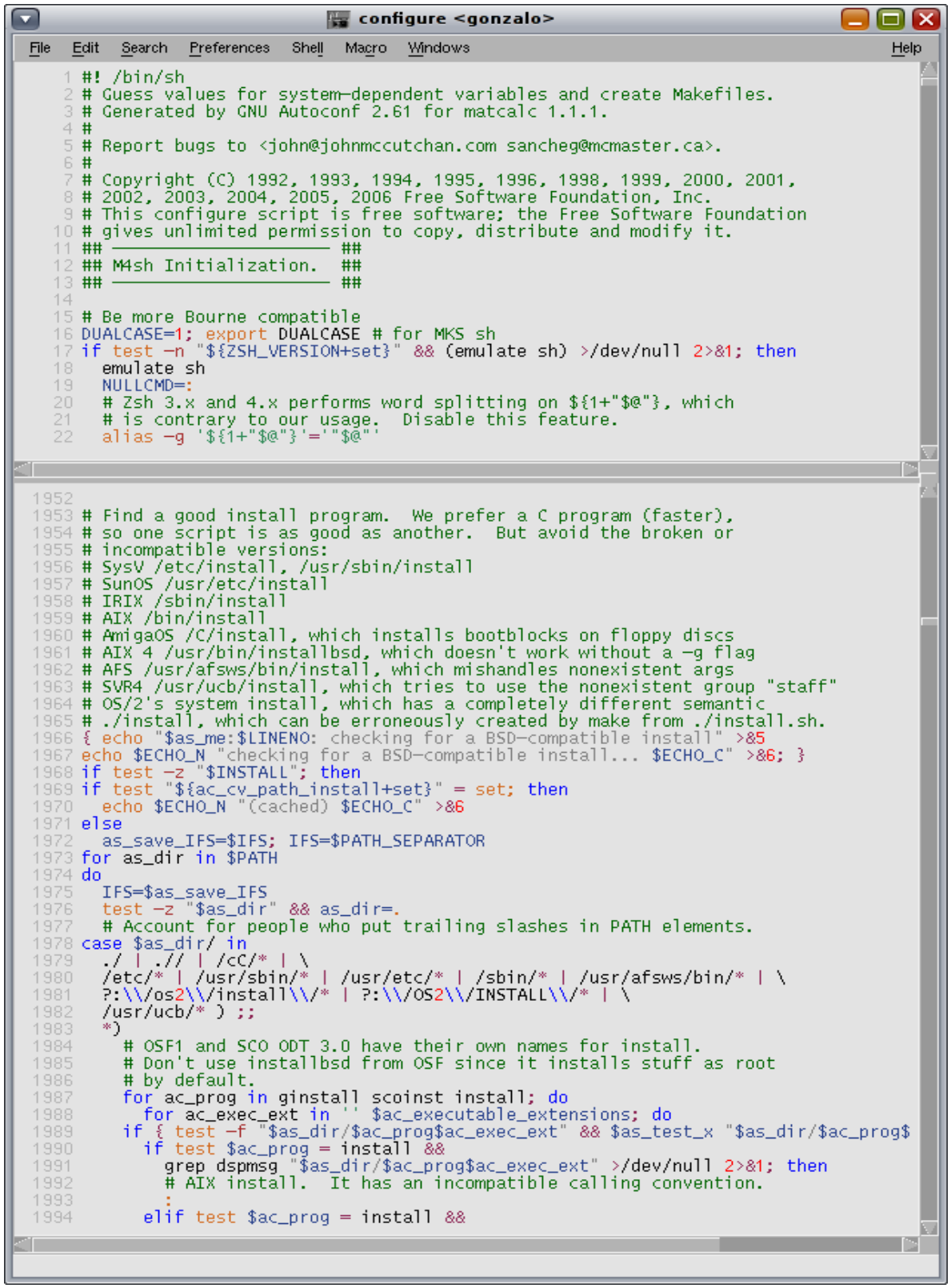

configure <gonzalo>
File Edit Search Preferences Shell Macro Windows Help
#! /bin/sh
Guess values for system-dependent variables and create Makefiles.
Generated by GNU Autoconf 2.61 for matcalc 1.1.1.
#
Report bugs to <john@johnmccutchan.com sancheg@mcmaster.ca>.
#
Copyright (C) 1992, 1993, 1994, 1995, 1996, 1998, 1999, 2000, 2001,
2002, 2003, 2004, 2005, 2006 Free Software Foundation, Inc.
This configure script is free software; the Free Software Foundation
gives unlimited permission to copy, distribute and modify it.
M4sh Initialization.
# Be more Bourne compatible
DUALCASE=1; export DUALCASE # for MKS sh
if test -n "${ZSH_VERSION+set}" && (emulate sh) >/dev/null 2>&1; then
emulate sh
NULLCMD=:
# Zsh 3.x and 4.x performs word splitting on ${1+"$@"}, which
# is contrary to our usage. Disable this feature.
alias -g '${1+"$@"}'='"$@"'
Find a good install program. We prefer a C program (faster),
so one script is as good as another. But avoid the broken or
incompatible versions:
SysV /etc/install, /usr/sbin/install
SunOS /usr/etc/install
IRIX /sbin/install
AIX /bin/install
AmigaOS /C/install, which installs bootblocks on floppy discs
AIX 4 /usr/bin/installbsd, which doesn't work without a -g flag
AFS /usr/afsws/bin/install, which mishandles nonexistent args
SVR4 /usr/ucb/install, which tries to use the nonexistent group "staff"
OS/2's system install, which has a completely different semantic
./install, which can be erroneously created by make from ./install.sh.
{ echo "$as_me:$LINENO: checking for a BSD-compatible install" >&5
echo $ECHO_N "checking for a BSD-compatible install... $ECHO_C" >&6; }
if test -z "$INSTALL"; then
if test "${ac_cv_path_install+set}" = set; then
echo $ECHO_N "(cached) $ECHO_C" >&6
else
as_save_IFS=$IFS; IFS=$PATH_SEPARATOR
for as_dir in $PATH
do
IFS=$as_save_IFS
test -z "$as_dir" && as_dir=.
Account for people who put trailing slashes in PATH elements.
case $as_dir/ in
./ | .// | /cC/* | \
/etc/* | /usr/sbin/* | /usr/etc/* | /sbin/* | /usr/afsws/bin/* | \
?:\\/os2\\/install\\/* | ?:\\/OS2\\/INSTALL\\/* | \
/usr/ucb/*) ;;
*)
# OSF1 and SCO ODT 3.0 have their own names for install.
# Don't use installbsd from OSF since it installs stuff as root
# by default.
for ac_prog in ginstall scoinst install; do
for ac_exec_ext in '' $ac_executable_extensions; do
if { test -f "$as_dir/$ac_prog$ac_exec_ext" && $as_test_x "$as_dir/ac_prog
if test $ac_prog = install &&
grep dspmsg "$as_dir/$ac_prog$ac_exec_ext" >/dev/null 2>&1; then
AIX install. It has an incompatible calling convention.
:
elif test $ac_prog = install &&

Fig. 32: `configure` script sample.

[6.2.1.3] Makefiles

A '**Makefile**' is a file that contains *build descriptions* (building instructions, or 'rules') that enable building a software package with the build tool, `make`. That is: it specifies how to derive the target program from each of its dependencies. Makefiles are parsed by the `make` tool, which then appropriately calls the compiler, assembler and (static) linking tools to actually compile, assemble and link the target program from each of the dependencies.

> **Note**: An overview of the compilation, assembly, and linking (static and dynamic) process is presented in section 9.14, p. 197

A Makefile consists of lines of text which define a file (or set of files) or a rule name as depending on a set of files. Output files are marked as depending on their source files, for example, and on files which they include internally, since they all affect the output. After each dependency is listed, a series of lines of tab-indented text may follow which define how to transform the input into the output, if the former has been modified more recently than the latter. In the case where such definitions are present, they are referred to as "build scripts" and are passed to the shell to generate the target file.

A makefile also can contain definitions of variables and inclusion of other makefiles. Variables in makefiles may be overridden in the command-line arguments passed to the make utility. This allows users to specify different behaviour for the build scripts and how to invoke programs, among other things. For example, the variable "CC" is frequently used in makefiles to refer to a C compiler, and the user may wish to provide an alternate compiler to use.

http://en.wikipedia.org/wiki/Makefile#Makefile_structure

Below is a very simple makefile that would compile a source called "helloworld.c" using cc, a C compiler, and specifies a "clean" target to remove the generated files, for example to start over. The .PHONY tag is a technicality that tells make that a particular target name does not produce an actual file. The `$@` and `$<` are two of the so-called internal macros (also known as automatic variables) and stand for the target name and "implicit" source, respectively. There are a number of other internal macros.

```
# Commands start with TAB not spaces
helloworld: helloworld.o
     cc -o $@ $<

helloworld.o: helloworld.c
     cc -c -o $@ $<

.PHONY: clean

clean:
     rm -f helloworld helloworld.o
```

http://en.wikipedia.org/wiki/Makefile#Example_makefile

Part of a *complex* Makefile – actually, the one generated by the `configure` script shown in Fig. 32, p. 107 – is shown in Fig. 33, p. 109.

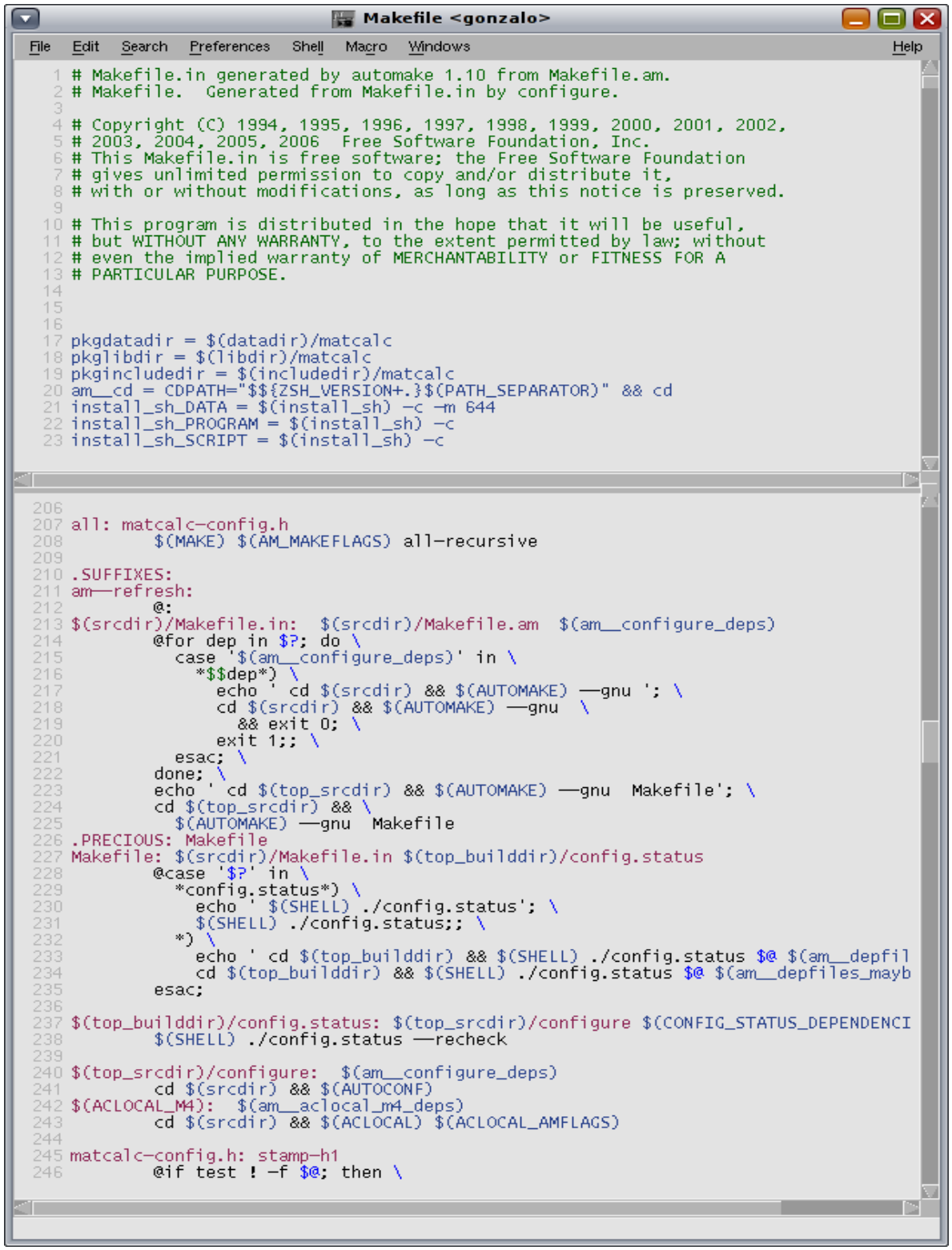

Makefile <gonzalo>
File Edit Search Preferences Shell Macro Windows Help
# Makefile.in generated by automake 1.10 from Makefile.am.
# Makefile. Generated from Makefile.in by configure.
# Copyright (C) 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002,
# 2003, 2004, 2005, 2006 Free Software Foundation, Inc.
# This Makefile.in is free software; the Free Software Foundation
# gives unlimited permission to copy and/or distribute it,
# with or without modifications, as long as this notice is preserved.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY, to the extent permitted by law; without
# even the implied warranty of MERCHANTABILITY or FITNESS FOR A
# PARTICULAR PURPOSE.
pkgdatadir = $(datadir)/matcalc
pkglibdir = $(libdir)/matcalc
pkgincludedir = $(includedir)/matcalc
am__cd = CDPATH="$${ZSH_VERSION+.}$(PATH_SEPARATOR)" && cd
install_sh_DATA = $(install_sh) -c -m 644
install_sh_PROGRAM = $(install_sh) -c
install_sh_SCRIPT = $(install_sh) -c
all: matcalc-config.h
$(MAKE) $(AM_MAKEFLAGS) all-recursive
.SUFFIXES:
am--refresh:
@:
$(srcdir)/Makefile.in: $(srcdir)/Makefile.am $(am__configure_deps)
@for dep in $?; do \
case '$(am__configure_deps)' in \
$$dep) \
echo ' cd $(srcdir) && $(AUTOMAKE) --gnu '; \
cd $(srcdir) && $(AUTOMAKE) --gnu \
&& exit 0; \
exit 1;; \
esac; \
done; \
echo ' cd $(top_srcdir) && $(AUTOMAKE) --gnu Makefile'; \
cd $(top_srcdir) && \
$(AUTOMAKE) --gnu Makefile
.PRECIOUS: Makefile
Makefile: $(srcdir)/Makefile.in $(top_builddir)/config.status
@case '$?' in \
config.status) \
echo ' $(SHELL) ./config.status'; \
$(SHELL) ./config.status;; \
*) \
echo ' cd $(top_builddir) && $(SHELL) ./config.status $@ $(am__depfil
cd $(top_builddir) && $(SHELL) ./config.status $@ $(am__depfiles_mayb
esac;
$(top_builddir)/config.status: $(top_srcdir)/configure $(CONFIG_STATUS_DEPENDENCI
$(SHELL) ./config.status --recheck
$(top_srcdir)/configure: $(am__configure_deps)
cd $(srcdir) && $(AUTOCONF)
$(ACLOCAL_M4): $(am__aclocal_m4_deps)
cd $(srcdir) && $(ACLOCAL) $(ACLOCAL_AMFLAGS)
matcalc-config.h: stamp-h1
@if test ! -f $@; then \

Fig. 33: `Makefile` example.

[6.2.1.4] Goals

The GNU build system has two goals ([GJ03], p. 1):

1. **Simplify the development of portable programs**.
2. **Simplify the building of programs that are distributed as source code**.

The first goal is achieved by the automatic generation of a `configure' shell script.

The second goal is achieved by the automatic generation of 'Makefiles' and other shell scripts that are typically used in the building process.

It must be noted that the GNU Build System is not devoid of problems:

> ***The GNU build system (GBS)***
>
> *(...) If GBS is so smart, why isn't everybody using it? One reason is that GBS is very Unix-centric and C/C++ dedicated. And even if you build C programs on Unix,* ***GBS only works effectively if your sourcecode complies with the GNU coding guidelines****.*
>
> *(...) Another reason why GBS is not used everywhere is the price it pays for its portability. For example,* ***the probes of autoconf are encoded as M4 macros. M4 is a powerful text processor, but its syntax is pretty low level and difficult****. Difficult enough to make this a major obstacle for extending the reach of GBS. While a godsend for less capable platforms, GBS has a much harder time winning the favor of developers on mainstream platforms. People simply don't want to give up their convenience (when writing probes) for the sake of some obscure platform no one's heard of.*
>
> *Another thorny issue is weak support for embedded software development and associated issues.* ***In the embedded world, everybody is cross-compiling; the build platform and the runtime platform are separate and very different in nature. In that case, automatic inspection of the build platform before the build will not get you very far****. In the embedded world, some kind of database holding the characteristics of the different platforms turns out better. The database is painful to maintain, but is the only solution that works. Autoconf has lately added some features to support cross-compilation.*
>
> ***Last but not least, one issue with GBS is the fact that it uses GNU Make and nothing other than GNU Make as a build tool. You will frequently have inconsistent builds, you will have a hard time debugging an inconsistent or a broken build, you will not be able to build a program called "clean" or "install", etc****. All these issues with GNU Make*[60] *undermine the important achievements of autoconf.* [61]
>
> http://freshmeat.net/articles/make-alternatives

[60] See, for example, the whitepaper "**What's Wrong With GNU `make`?**" at: http://www.conifersystems.com/whitepapers/gnu-make/

[61] The boldface type used here is for 'highlighting' passages of the quote, but is *not* present in the original text.

[6.2.1.5] User's perspective

From the *user's* perspective, the utility of the GNU build system is addressed by the aforementioned second goal: the simplification it provides when it comes to building software distributed as *source code* (as opposed to a binary distribution of a package) which squarely addresses the *usability* quality of a software package.

The typical commands an end user has to perform to build a software package, which has been prepared using the GNU Build System, are commented below:

> *When you download an autoconfiguring package, it usually has a filename like: `foo-1.0.tar.gz' where the number is a version number. To install it, first you have to unpack the package to a directory someplace:*
>
> ```
> % gunzip foo-1.0.tar.gz
> % tar xf foo-1.0.tar
> ```
>
> *Then you enter the directory and look for files like `README' or `INSTALL' that explain what you need to do. Almost always this amounts to typing the following commands:*
>
> ```
> % cd foo-1.0
> % ./configure
> % make
> % make check
> % su
> # make install
> ```
>
> *The `configure' command invokes a shell script that is distributed with the package that configures the package for you automatically. First it probes your system through a set of tests that allow it to determine things it needs to know, and then it uses this knowledge to generate automatically a `Makefile' from a template stored in a file called `Makefile.in'. When you invoke `make' with no argument, it executes the default target of the generated `Makefile'. That target will compile your source code, but will not install it. If your software comes with self-tests then you can compile and run them by typing `make check'. To install your software, you need to explicitly invoke `make' again with the target `install'.*
>
> ([GJ03], p. 9)

The aforementioned commands assume that the user has a working compiling environment on his/her platform (the `make` tool, and a properly installed compiler suite like the GNU Compiler Collection, GCC, invoked with the command `gcc`) and the unpacking tools (`gunzip` and `tar`). These tools are standard in any Linux distribution, and are available for MacOS X and Solaris/OpenSolaris platforms as well[62].

[62] Solaris/OpenSolaris platforms can also include SunStudio compiler suite, that could be used in lieu of GCC; however, programs build around the GNU Build System expect GCC, so some changes *might* be required to use other compiling suite. Additionally, Solaris `make` command is not compatible with GNU's `make` command, and the latter is renamed `gmake` in Solaris platforms (but not in OpenSolaris). Since the GNU Build Systems expects `make`, depending on the platform, the user might have to ensure the GNU tools are found first in the command search path, and eventually provide an appropriate symbolic link for `gmake` named `make`.

[6.2.1.6] Developer's perspective

From the *developer's* perspective, apart from the same usability improvements provided for the end user, the other underlying qualities addressed by the GNU build system are clearly that of *maintainability*, *portability, verifiability*, and *reliability*:

> *Some tasks that are simplified by the GNU build system include:*
>
> - *Building multidirectory software packages. It is much more difficult to use raw* `make` *recursively. Having simplified this step, the developer is encouraged to organize his source code in a deep directory tree rather than lump everything under the same directory. Developers that use raw* `make` *often can't justify the inconvenience of recursive make and prefer to disorganize their source code. With the GNU tools this is no longer necessary.*
> - *Automatic configuration. You will never have to tell your users that they need to edit your Makefile. You yourself will not have to edit your Makefiles as you move new versions of your code back and forth between different machines.*
> - *Automatic makefile generation. Writing makefiles involves a lot of repetition, and in large projects it will get on your nerves. The GNU build system instead requires you to write* `` `Makefile.am' `` *files that are much more terse and easy to maintain.*
> - *Support for test suites. You can very easily write test suite code, and by adding one extra line in your* `` `Makefile.am' `` *make a check target available such that you can compile and run the entire test suite by running* `make check`*.*
> - *Automatic distribution building. The GNU build tools are meant to be used in the development of free software, therefore if you have a working build system in place for your programs, you can create a source code distribution out of it by running* `make distcheck`*.*
> - *Shared libraries. Building shared libraries becomes as easy as building static libraries.*
>
> ([GJ03], p. 2)

The typical steps a *developer* has to perform to prepare a software package using the GNU Build System, are the following:

1. Prepare the files `configure.ac` and `Makefile.am` with appropriate instructions for the `autoconf` and `automake` tools
2. Execute the following tools[63]:

```
$ aclocal -I m4
$ autoheader
$ automake --add-missing –gnu --foreign
$ autoconf
```

[63] Alternatively, the utility `autoreconf` repeatedly runs `autoconf`, `autoheader`, `aclocal`, `automake`, and other tools to update the GNU build system in the specified directories and their subdirectories.

Note: eventually, the `autoscan` tool should be used by the developer *prior* to running the `aclocal`, `autoheader`, `automake` and `autoconf` tools. The `autoscan` utility searches the source files for common portability problems, checks for incompleteness of the file `configure.ac`, and creates a file `configure.scan` which is a preliminary `configure.ac` for the software package.

The above steps will produce the `configure` shell script, as intended for the end user. Therefore, as the end user would do, the steps required to build the software package from the `configure` script are typically the following:

```
$ ./configure
$ make
```

If the software package contains tests, these can be compiled and run with the following command:

```
$ make check
```

Installing the software once it has been compiled (and eventually tested) requires the following commands[64]:

```
$ su
# make install
```

To check at build time any specific options the software package might support, the `configure` shell script can be queried with the following command:

```
$ ./configure –help
```

The diagram in Fig. 34 shows the relationship between the key GNU build system files and tools, and their usage by the corresponding stakeholders. Note that the end user does *not* require to have the 'Autotools' installed in his/her system, but just a working compiling environment, which amounts to the `make` tool and a compiler/assembler/linker of choice (typically, if the GNU `make` command and GNU `gcc` compiler suite are available, it is assured to work with the GNU build system, and has probably been tested by the developer, but the user is *not* necessarily limited to that tool-chain; see note 62).

[64] Assuming administrator privileges for installation is only necessary if the developer desires to install the software in a system location, as opposed to performing an installation under *e.g.* his/her 'home' folder.

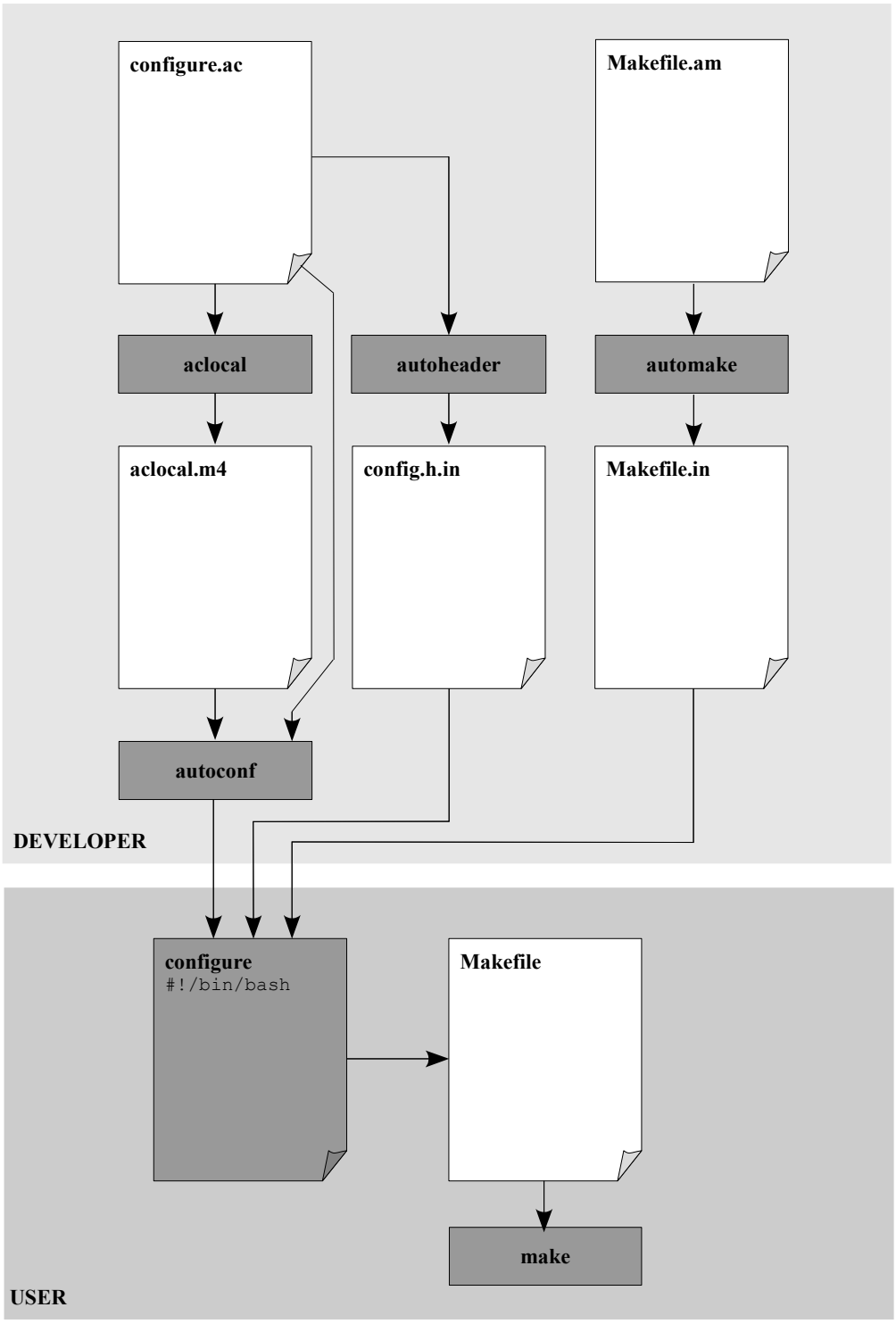


Fig. 34: Key GNU build system tools and files, from a developer and a user perspective.

[6.3] Doxygen

Doxygen is a tool for generating formatted, browsable, and printable documentation from specially-formatted comment blocks in source code. This allows for developer documentation to be embedded in the files where it is most likely to be kept complete and up-to-date.

http://old.lugatgt.org/articles/doxygen/

Doxygen is a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors), Fortran, VHDL, PHP, C#, and to some extent D.

It can help you in three ways:

(1) It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.

(2) You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. You can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

(3) You can even `abuse' doxygen for creating normal documentation.

http://www.stack.nl/~dimitri/doxygen/.

Doxygen is a command-line utility[65]. To generate the documentation for a project, these steps are typically followed:

1. Document the source code with special documentation blocks[66]:

```
/** \file
    \brief FEM_test: CLI tool for testing MatCalc's FEM algorithm.
*/
```

2. Generate a configuration file by calling Doxygen with the `-g` option:

```
$ doxygen -g <config_file>
```

3. Edit the configuration file `<config_file>` so it matches the project:

```
# The PROJECT_NAME tag is a single word (or a sequence of words surrounded
# by quotes) that should identify the project.

PROJECT_NAME           = MatCalc

# The OUTPUT_DIRECTORY tag is used to specify the (relative or absolute)
```

[65] Calling doxygen with the `--help` option at the command line will give you a brief description of the usage of the program.

[66] That is, comment the code using doxygen supported comment format, so that it will be appropriately captured and categorized when the source file is parsed.

```
# base path where the generated documentation will be put.

OUTPUT_DIRECTORY       = ../doc
```

4. Call Doxygen to generate the documentation, based on the settings in the configuration file:

```
$ doxygen <config_file>
```

Doxygen features include (http://www.stack.nl/~dimitri/doxygen/features.html):

- Support for C/C++, Java, (Corba and Microsoft) Java, Python, IDL, C#, Objective-C and to some extent D and PHP sources.
- Supports documentation of files, namespaces, packages, classes, structs, unions, templates, variables, functions, typedefs, enums and defines.
- Automatically generates class and collaboration diagrams in HTML (as clickable image maps) and LaTeX (as Encapsulated PostScript images).
- Uses the `dot` tool of the Graphviz tool kit to generate include dependency graphs, collaboration diagrams, call graphs, directory structure graphs, and graphical class hierarchy graphs.
- Flexible comment placement: Allows you to put documentation in the header file (before the declaration of an entity), source file (before the definition of an entity) or in a separate file.
- Generates a list of all members of a class (including any inherited members) along with their protection level.
- Outputs documentation in on-line format (HTML and UNIX man page) and off-line format (LaTeXand RTF) simultaneously (any of these can be disabled if desired). All formats are optimized for ease of reading.
- Includes a full C preprocessor to allow proper parsing of conditional code fragments and to allow expansion of all or part of macros definitions.
- Automatically detects public, protected and private sections. Extraction of private class members is optional.
- Automatically generates references to documented classes, files, namespaces and members. Documentation of global functions, global variables, typedefs, defines and enumerations is also supported.
- References to base/super classes and inherited/overridden members are generated automatically.
- Includes a fast, rank based search engine to search for strings or words in the class and member documentation.

- Allows references to documentation generated for other projects (or another part of the same project) in a location independent way.
- Allows inclusion of source code examples that are automatically cross-referenced with the documentation.
- Inclusion of undocumented classes is also supported, allowing to quickly learn the structure and interfaces of a (large) piece of code without looking into the implementation details.
- Allows automatic cross-referencing of (documented) entities with their definition in the source code.
- All source code fragments are syntax highlighted for ease of reading.
- Allows inclusion of function/member/class definitions in the documentation.
- All options are read from an easy to edit and (optionally) annotated configuration file.
- Documentation and search engine can be transferred to another location or machine without regenerating the documentation.

An example of the graphical output capabilities and syntax highlighting of Doxygen is provided by Doxygen's graph legend page for C++ classes (Fig. 35):

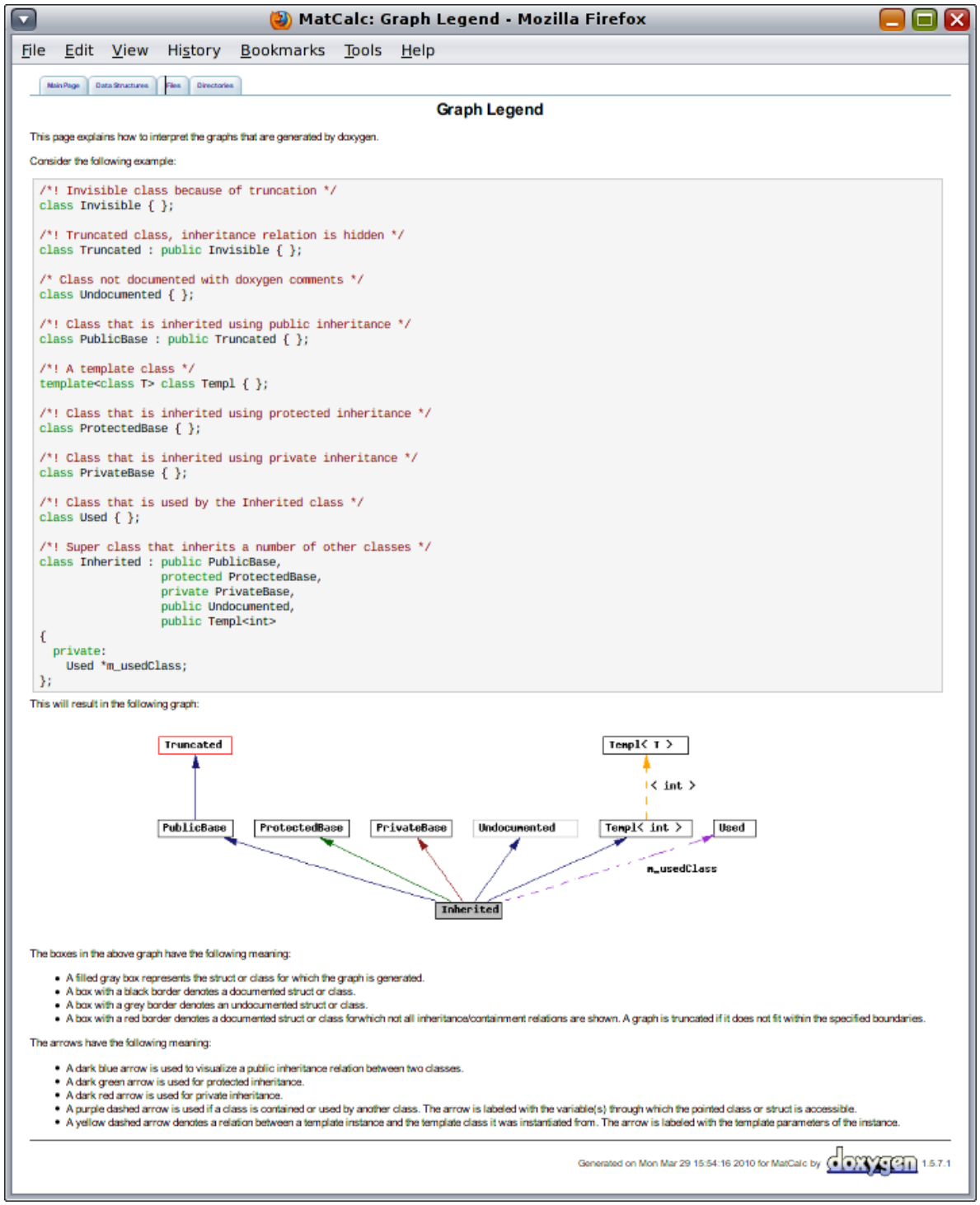

MatCalc: Graph Legend - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Graph Legend
This page explains how to interpret the graphs that are generated by doxygen.
Consider the following example:
/*! Invisible class because of truncation */
class Invisible { };
/*! Truncated class, inheritance relation is hidden */
class Truncated : public Invisible { };
/* Class not documented with doxygen comments */
class Undocumented { };
/*! Class that is inherited using public inheritance */
class PublicBase : public Truncated { };
/*! A template class */
template<class T> class Templ { };
/*! Class that is inherited using protected inheritance */
class ProtectedBase { };
/*! Class that is inherited using private inheritance */
class PrivateBase { };
/*! Class that is used by the Inherited class */
class Used { };
/*! Super class that inherits a number of other classes */
class Inherited : public PublicBase,
protected ProtectedBase,
private PrivateBase,
public Undocumented,
public Templ<int>
{
private:
Used *m_usedClass;
};
This will result in the following graph:
Truncated
Templ< T >
< int >
PublicBase
ProtectedBase
PrivateBase
Undocumented
Templ< int >
Used
m_usedClass
Inherited
The boxes in the above graph have the following meaning:
A filled gray box represents the struct or class for which the graph is generated.
A box with a black border denotes a documented struct or class.
A box with a grey border denotes an undocumented struct or class.
A box with a red border denotes a documented struct or class forwhich not all inheritance/containment relations are shown. A graph is truncated if it does not fit within the specified boundaries.
The arrows have the following meaning:
A dark blue arrow is used to visualize a public inheritance relation between two classes.
A dark green arrow is used for protected inheritance.
A dark red arrow is used for private inheritance.
A purple dashed arrow is used if a class is contained or used by another class. The arrow is labeled with the variable(s) through which the pointed class or struct is accessible.
A yellow dashed arrow denotes a relation between a template instance and the template class it was instantiated from. The arrow is labeled with the template parameters of the instance.
Generated on Mon Mar 29 15:54:16 2010 for MatCalc by doxygen 1.5.7.1

Fig. 35: Doxygen graph legend.

An example of automatic cross-referencing of entities and automatic call and caller graphs generation is provided in Fig. 36 for the `read_matcalc_file()` function:

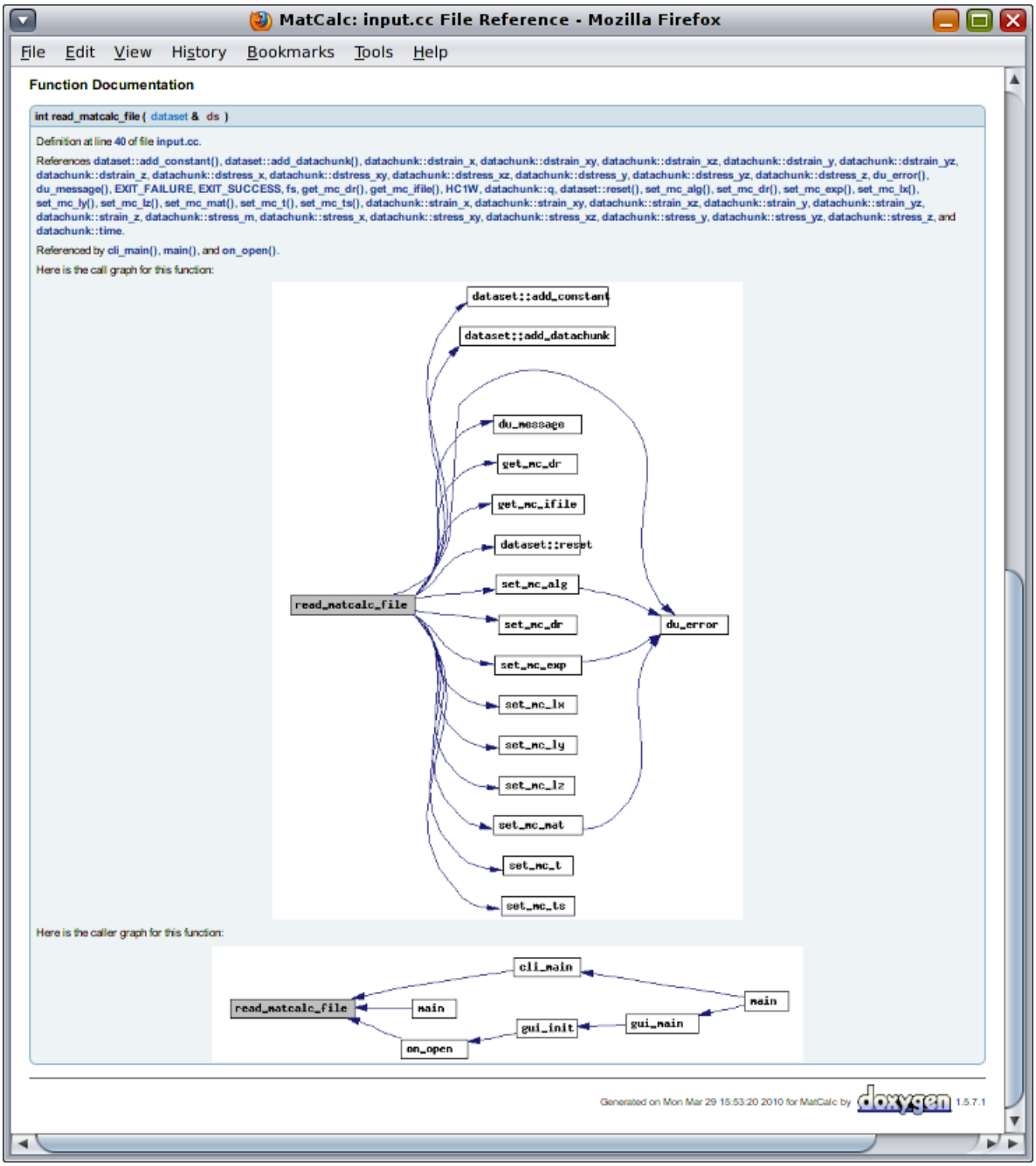

MatCalc: input.cc File Reference - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Function Documentation
int read_matcalc_file (dataset & ds)
Definition at line 40 of file input.cc.
References dataset::add_constant(), dataset::add_datachunk(), datachunk::dstrain_x, datachunk::dstrain_xy, datachunk::dstrain_xz, datachunk::dstrain_y, datachunk::dstrain_yz, datachunk::dstrain_z, datachunk::dstress_x, datachunk::dstress_xy, datachunk::dstress_xz, datachunk::dstress_y, datachunk::dstress_yz, datachunk::dstress_z, du_error(), du_message(), EXIT_FAILURE, EXIT_SUCCESS, fs, get_mc_dr(), get_mc_ifile(), HC1W, datachunk::q, dataset::reset(), set_mc_alg(), set_mc_dr(), set_mc_exp(), set_mc_lx(), set_mc_ly(), set_mc_lz(), set_mc_mat(), set_mc_t(), set_mc_ts(), datachunk::strain_x, datachunk::strain_xy, datachunk::strain_xz, datachunk::strain_y, datachunk::strain_yz, datachunk::strain_z, datachunk::stress_m, datachunk::stress_x, datachunk::stress_xy, datachunk::stress_xz, datachunk::stress_y, datachunk::stress_yz, datachunk::stress_z, and datachunk::time.
Referenced by cli_main(), main(), and on_open().
Here is the call graph for this function:
dataset::add_constant
dataset::add_datachunk
du_message
get_mc_dr
get_mc_ifile
dataset::reset
set_mc_alg
read_matcalc_file
set_mc_dr
du_error
set_mc_exp
set_mc_lx
set_mc_ly
set_mc_lz
set_mc_mat
set_mc_t
set_mc_ts
Here is the caller graph for this function:
cli_main
read_matcalc_file
main
main
gui_init
gui_main
on_open
Generated on Mon Mar 29 15:53:20 2010 for MatCalc by doxygen 1.5.7.1

Fig. 36: `input.cc` call and caller graphs, generated by Doxygen.

All entities listed under the ‘References’ and ‘Referenced by’ labels, as well as all entities in the ‘call’ and ‘caller’ graphs are hyper-links to the corresponding sections of the Doxygen documentation.

An example of the 'brief' comment (under 'matcalc.cc File Reference' heading) and the 'detailed' comment (under the 'Detailed Description' heading) in the source code extracted by Doxygen, and posted to the File Reference for the file `matcalc.cc`, alongside the `#include` dependency graph for the file, is shown in Fig. 37:

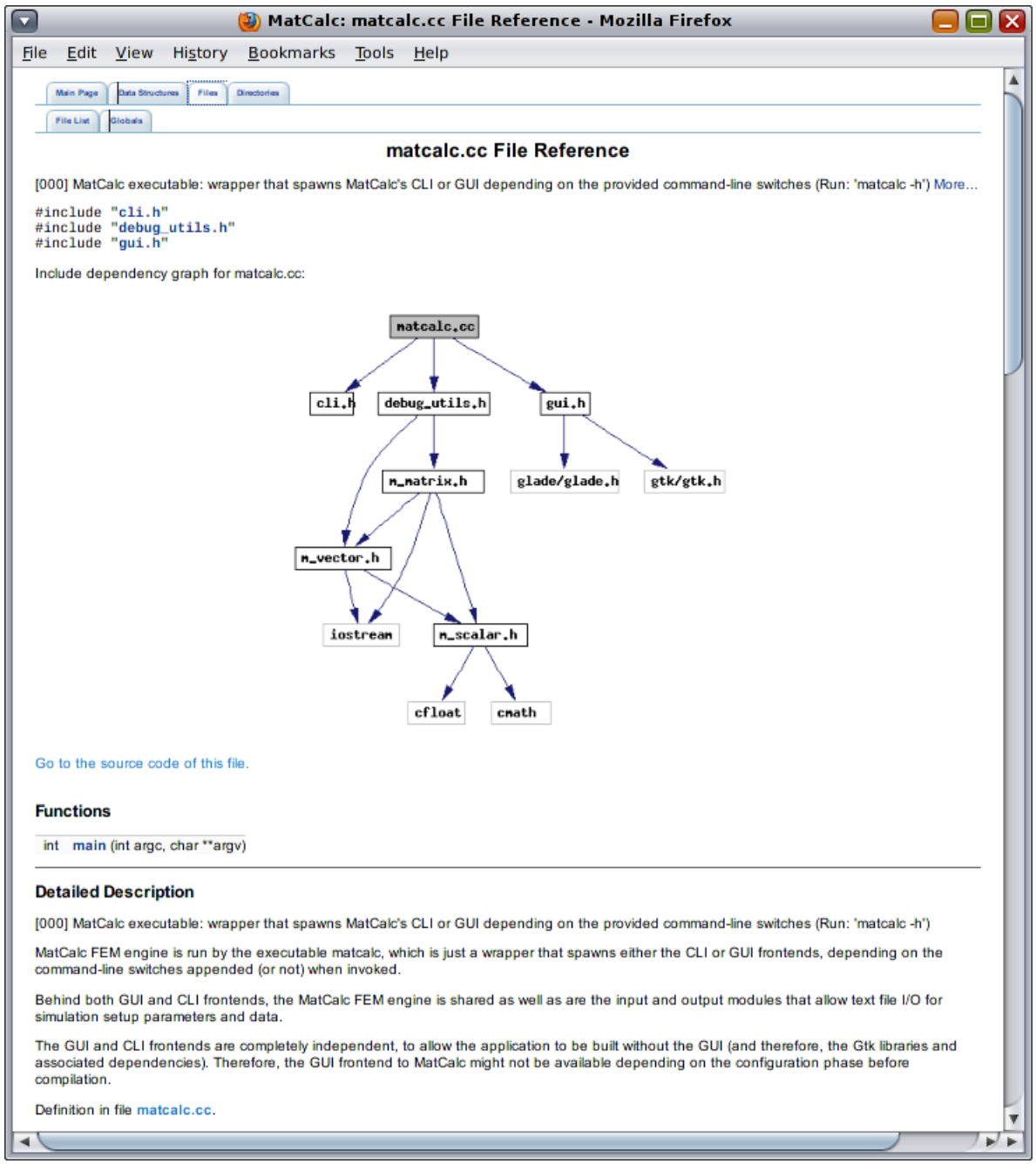

MatCalc: matcalc.cc File Reference - Mozilla Firefox
File Edit View History Bookmarks Tools Help
matcalc.cc File Reference
[000] MatCalc executable: wrapper that spawns MatCalc's CLI or GUI depending on the provided command-line switches (Run: 'matcalc -h') More...
#include "cli.h"
#include "debug_utils.h"
#include "gui.h"
Include dependency graph for matcalc.cc:
matcalc.cc
cli.h
debug_utils.h
gui.h
m_matrix.h
glade/glade.h
gtk/gtk.h
m_vector.h
iostream
m_scalar.h
cfloat
cmath
Go to the source code of this file.
Functions
int main (int argc, char **argv)
Detailed Description
[000] MatCalc executable: wrapper that spawns MatCalc's CLI or GUI depending on the provided command-line switches (Run: 'matcalc -h')
MatCalc FEM engine is run by the executable matcalc, which is just a wrapper that spawns either the CLI or GUI frontends, depending on the command-line switches appended (or not) when invoked.
Behind both GUI and CLI frontends, the MatCalc FEM engine is shared as well as are the input and output modules that allow text file I/O for simulation setup parameters and data.
The GUI and CLI frontends are completely independent, to allow the application to be built without the GUI (and therefore, the Gtk libraries and associated dependencies). Therefore, the GUI frontend to MatCalc might not be available depending on the configuration phase before compilation.
Definition in file matcalc.cc.

Fig. 37: Brief / detailed comments and `#include` dependency graph for `matcalc.cc`, generated by Doxygen.

An example of syntax highlighted source code fragment, and automatic cross-referencing of entities with their definition therein, is given in Fig. 38 for the file `matcalc.cc`:

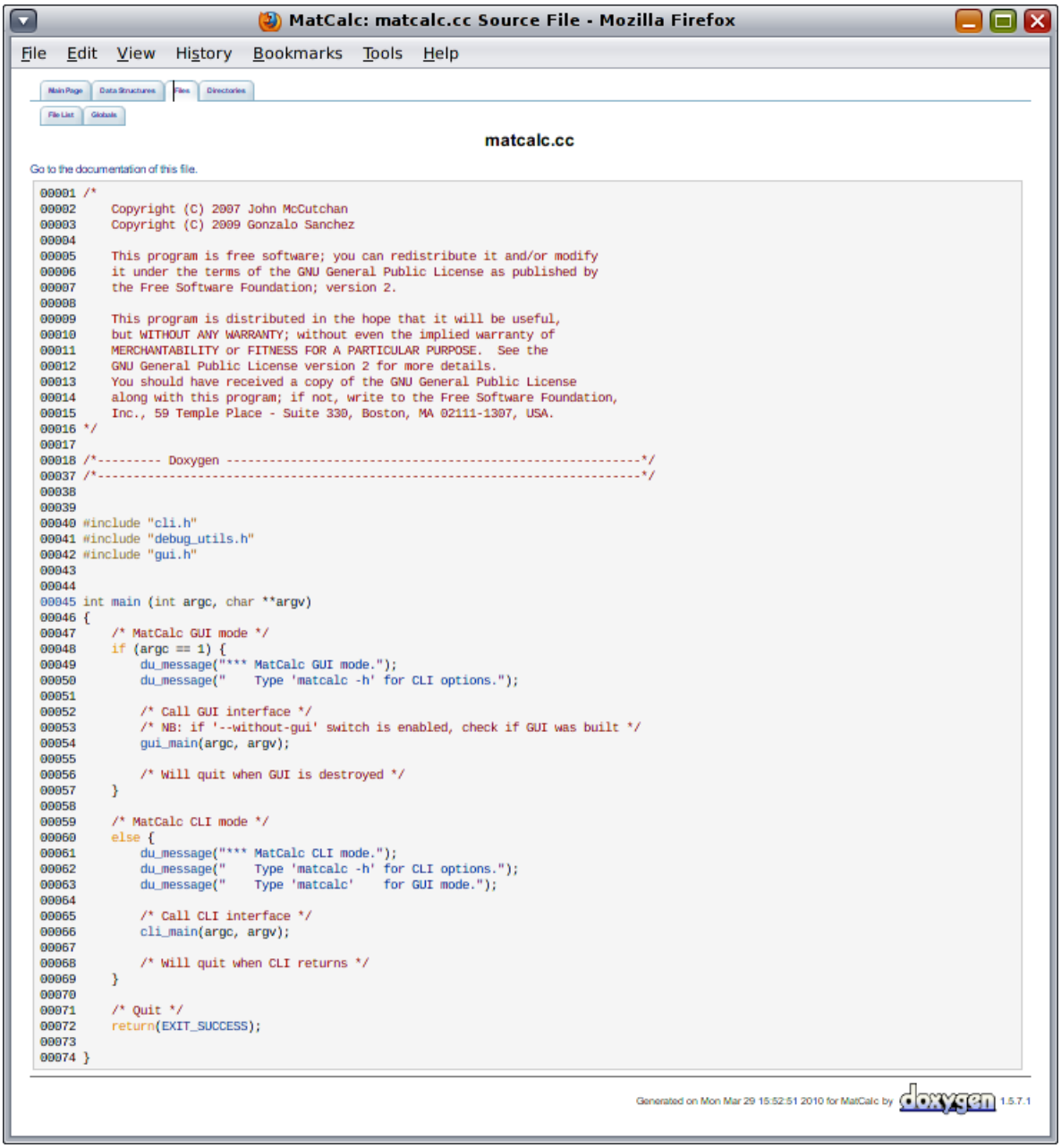

MatCalc: matcalc.cc Source File - Mozilla Firefox
File Edit View History Bookmarks Tools Help
matcalc.cc
Go to the documentation of this file.
00001 /*
00002 Copyright (C) 2007 John McCutchan
00003 Copyright (C) 2009 Gonzalo Sanchez
00004
00005 This program is free software; you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation; version 2.
00008
00009 This program is distributed in the hope that it will be useful,
00010 but WITHOUT ANY WARRANTY; without even the implied warranty of
00011 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012 GNU General Public License version 2 for more details.
00013 You should have received a copy of the GNU General Public License
00014 along with this program; if not, write to the Free Software Foundation,
00015 Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
00016 */
00017
00018 /*--------- Doxygen ----------*/
00037 /*----------*/
00038
00039
00040 #include "cli.h"
00041 #include "debug_utils.h"
00042 #include "gui.h"
00043
00044
00045 int main (int argc, char **argv)
00046 {
00047 /* MatCalc GUI mode */
00048 if (argc == 1) {
00049 du_message("*** MatCalc GUI mode.");
00050 du_message(" Type 'matcalc -h' for CLI options.");
00051
00052 /* Call GUI interface */
00053 /* NB: if '--without-gui' switch is enabled, check if GUI was built */
00054 gui_main(argc, argv);
00055
00056 /* Will quit when GUI is destroyed */
00057 }
00058
00059 /* MatCalc CLI mode */
00060 else {
00061 du_message("*** MatCalc CLI mode.");
00062 du_message(" Type 'matcalc -h' for CLI options.");
00063 du_message(" Type 'matcalc' for GUI mode.");
00064
00065 /* Call CLI interface */
00066 cli_main(argc, argv);
00067
00068 /* Will quit when CLI returns */
00069 }
00070
00071 /* Quit */
00072 return(EXIT_SUCCESS);
00073
00074 }
Generated on Mon Mar 29 15:52:51 2010 for MatCalc by doxygen 1.5.7.1

Fig. 38: `matcalc.cc` code listing, generated by Doxygen.

Note the missing lines between line 18 and 37; these correspond to the ‘brief’ and ‘detailed’ comments targeted to Doxygen parser, which are excluded in the code listing and posted to the corresponding File Reference (Fig. 37).

The rationale for using an automated documentation system is mainly that of, precisely, *automation*:

> ***Why use an automated system?***
>
> ***Documentation is up-to-date****: You are much more likely to change a comment atop of the function you fiddle with, than fire up MS Word, look for the function documentation, and change it there.*
>
> ***Reuse of your own comments****: Assuming you already discovered the virtue of commenting your own code, the market value of your comments just doubled.*
>
> ***Automatic formatting, and cross-linking****: With a few simple markups, you get documentation that looks professional and consistent, and creates links to the description of all important entities. Forget about struggling with MS Word.*
>
> ***In-code comments carry important meta information****: Not all important information is available from the actual "raw" source code. Interface contracts, pre- and post-conditions, side effects, meaning of special parameters, etc. need to be known to anyone who is to maintain the code - be it yourself or someone else. A formal style for these comments, in conjunction with a parser (like doxygen's XML export) can make this information available in customized format for various recipients: be it project management, testers, and the like.*
>
> http://www.codeproject.com/KB/tips/doxysetup.aspx#rationale

[6.4] Valgrind

Valgrind [Sew00] is a binary instrumentation framework for memory debugging, memory leak detection, and profiling:

> *Valgrind is an award-winning instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.*
>
> *The Valgrind distribution currently includes six production-quality tools: a memory error detector, two thread error detectors, a cache and branch-prediction profiler, a call-graph generating cache profiler, and a heap profiler. It also includes two experimental tools: a heap/stack/global array overrun detector, and a SimPoint basic block vector generator. It runs on the following platforms: X86/Linux, AMD64/Linux, PPC32/Linux, PPC64/Linux, and X86/Darwin (Mac OS X).*
>
> http://valgrind.org/
>
> ***Why should you use Valgrind?***[67]
>
> ***Valgrind is easy to use**. Valgrind uses dynamic binary instrumentation, so you don't need to modify, recompile or relink your applications. Just prefix your command line with valgrind and everything works.*
>
> ***Valgrind works with programs written in any language**. Because Valgrind works directly with program binaries, it works with programs written in any programming language, be they compiled, just-in-time compiled, or interpreted.*
>
> ***Valgrind gives 100% coverage of user-space code, even within system libraries**. You can even use Valgrind on programs for which you don't have the source code.*
>
> ***Valgrind will save you hours of debugging time**. With Valgrind tools you can automatically detect many memory management and threading bugs. This gives you confidence that your programs are free of many common bugs, some of which would take hours to find manually, or never be found at all. You can find and eliminate bugs before they become a problem.*
>
> ***Valgrind can help you speed up your programs**. With Valgrind tools you can also perform very detailed profiling to help find bottlenecks in your programs.*
>
> http://valgrind.org/info/about.html

Valgrind's utility comes with a price: since it implements a software CPU underneath the binary code being analyzed, the programs run significantly slower than when run by the hardware CPU.

[67] The boldface type applied here if for 'highlighting' sections of particular relevance in the quote; it is *not* part of the original text.

When should you use Valgrind?[68]

All the time. *For small programs with short run-times, when developing you can always run the program under a Valgrind tool (usually Memcheck), knowing that memory bugs will be found immediately.*

In automatic testing. *By using Valgrind tools in your automatic unit, integration, system, or regression test, you can be confident no code will be unchecked.*

After big changes. *To ensure new bugs haven't been introduced in the new code.*

When a bug occurs. *Get instant feedback about what the bug is, where it occurred, and why.*

When a bug is suspected. *Is your program behaving oddly? Use a Valgrind tool to discover if a bug is the cause.*

As for Valgrind's profiling tools, use those whenever you want information about how your program is spending its time, or you want to speed it up.

http://valgrind.org/info/about.html

Using Valgrind for memory checking purposes is absolutely straightforward. A typical invocation would be[69]:

```
$ valgrind -v –memcheck:leak-check=yes –log-file-exactly=logfile ./prog
```

A program devoid of memory leaks, would produce results similar to the following (corresponding to the `dset_diff` tool for MatCalc; see section 5.3.8.2, p. 96):

```
==17925== Memcheck, a memory error detector.
==17925== Copyright (C) 2002-2006, and GNU GPL'd, by Julian Seward et al.
==17925== Using LibVEX rev 1658, a library for dynamic binary translation.
==17925== Copyright (C) 2004-2006, and GNU GPL'd, by OpenWorks LLP.
==17925== Using valgrind-3.2.1, a dynamic binary instrumentation framework.
==17925== Copyright (C) 2000-2006, and GNU GPL'd, by Julian Seward et al.
==17925==
==17925== My PID = 17925, parent PID = 29514.  Prog and args are:
==17925==    ./dset_diff
==17925==    dset1.dat
==17925==    dset2.dat
==17925==
--17925--
--17925-- Command line
--17925--    ./dset_diff
--17925--    dset1.dat
--17925--    dset2.dat
--17925-- Startup, with flags:
--17925--    -v
--17925--    --tool=memcheck
--17925--    --leak-check=yes
--17925--    --show-reachable=yes
--17925--    --log-file=/u0/grad/sancheg/Out/valgrind.log
--17925-- Contents of /proc/version:
--17925--   Linux version 2.6.18-164.9.1.el5 (mockbuild@x86-005.build.bos.redhat.com)
(gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)) #1 SMP Wed Dec 9 03:27:37 EST 2009
```

[68] The boldface type applied here if for 'highlighting' sections of particular relevance in the quote; it is *not* part of the original text.

[69] The shell script listed in section9.10 (p. 184) provides several and very similar examples.

```
--17925-- Arch and hwcaps: AMD64, amd64-sse2
--17925-- Valgrind library directory: /usr/lib64/valgrind
--17925-- Reading syms from
/nfs/u0/grad/sancheg/Projects/virtlab/svn/1.1/matcalc/src/dset_diff (0x400000)
--17925-- Reading syms from /usr/lib64/valgrind/amd64-linux/memcheck (0x38000000)
--17925--    object doesn't have a dynamic symbol table
--17925-- Reading syms from /lib64/ld-2.5.so (0x383CA00000)
--17925-- Reading suppressions file: /usr/lib64/valgrind/default.supp
--17925-- Reading syms from /usr/lib64/valgrind/amd64-linux/vgpreload_core.so
(0x4802000)
--17925-- Reading syms from /usr/lib64/valgrind/amd64-linux/vgpreload_memcheck.so
(0x4A03000)
--17925-- REDIR: 0x383CA14440 (index) redirected to 0x4A06550 (index)
--17925-- REDIR: 0x383CA145F0 (strcmp) redirected to 0x4A067D0 (strcmp)
--17925-- REDIR: 0x383CA14620 (strlen) redirected to 0x4A06700 (strlen)
--17925-- Reading syms from /usr/lib64/libgtk-x11-2.0.so.0.1000.4 (0x3847A00000)
--17925--    object doesn't have a symbol table
--17925-- Reading syms from /usr/lib64/libgdk-x11-2.0.so.0.1000.4 (0x384A000000)
--17925--    object doesn't have a symbol table

                                (several lines suppressed here)

==17925== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 1)
--17925--
--17925-- supp:    4 Fedora-Core-6-hack3-ld25
==17925== malloc/free: in use at exit: 0 bytes in 0 blocks.
==17925== malloc/free: 5,461 allocs, 5,461 frees, 454,809 bytes allocated.
==17925==
==17925== All heap blocks were freed -- no leaks are possible.
--17925--  memcheck: sanity checks: 39 cheap, 2 expensive
--17925--  memcheck: auxmaps: 1381 auxmap entries (88384k, 86M) in use
--17925--  memcheck: auxmaps: 6016040 searches, 57284561 comparisons
--17925--  memcheck: SMs: n_issued      = 64 (1024k, 1M)
--17925--  memcheck: SMs: n_deissued    = 0 (0k, 0M)
--17925--  memcheck: SMs: max_noaccess  = 524287 (8388592k, 8191M)
--17925--  memcheck: SMs: max_undefined = 0 (0k, 0M)
--17925--  memcheck: SMs: max_defined   = 1452 (23232k, 22M)
--17925--  memcheck: SMs: max_non_DSM   = 64 (1024k, 1M)
--17925--  memcheck: max sec V bit nodes:    0 (0k, 0M)
--17925--  memcheck: set_sec_vbits8 calls: 0 (new: 0, updates: 0)
--17925--  memcheck: max shadow mem size:   5168k, 5M
--17925-- translate:            fast SP updates identified: 5,456 ( 88.3%)
--17925-- translate:   generic_known SP updates identified: 387 (  6.2%)
--17925-- translate: generic_unknown SP updates identified: 333 (  5.3%)
--17925--     tt/tc: 10,544 tt lookups requiring 10,978 probes
--17925--     tt/tc: 10,544 fast-cache updates, 5 flushes
--17925--  transtab: new        4,802 (122,280 -> 2,217,188; ratio 181:10) [0 scs]
--17925--  transtab: dumped     0 (0 -> ??)
--17925--  transtab: discarded  14 (295 -> ??)
--17925-- scheduler: 3,961,401 jumps (bb entries).
--17925-- scheduler: 39/17,156 major/minor sched events.
--17925--    sanity: 40 cheap, 2 expensive checks.
--17925--    exectx: 30,011 lists, 292 contexts (avg 0 per list)
--17925--    exectx: 10,926 searches, 86,644 full compares (7,930 per 1000)
--17925--    exectx: 0 cmp2, 6 cmp4, 0 cmpAll
```

Amid the copious information produced (the flag `-v` was specified upon invocation, which causes system libraries information to be included in the log, as well as statistics pertaining to Valgrind's binary instrumentation) Valgrind's report indicates the program has performed a 'clean' memory operation, freeing all allocated memory upon exit and thus producing no memory leaks.

Memory that has not been freed by the system upon program termination is marked as 'still reachable', like in the following case (corresponding to the `lists` tool for MatCalc; see section 5.3.8.1, p. 95):

```
==32057== Memcheck, a memory error detector.
==32057== Copyright (C) 2002-2006, and GNU GPL'd, by Julian Seward et al.
==32057== Using LibVEX rev 1658, a library for dynamic binary translation.
==32057== Copyright (C) 2004-2006, and GNU GPL'd, by OpenWorks LLP.
==32057== Using valgrind-3.2.1, a dynamic binary instrumentation framework.
==32057== Copyright (C) 2000-2006, and GNU GPL'd, by Julian Seward et al.
==32057==
==32057== My PID = 32057, parent PID = 29514.  Prog and args are:
==32057==    ./lists

                                    (several lines suppressed here)

==32057== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 1)
--32057--
--32057-- supp:    4 Fedora-Core-6-hack3-ld25
==32057== malloc/free: in use at exit: 816,531 bytes in 9,475 blocks.
==32057== malloc/free: 10,114 allocs, 639 frees, 884,066 bytes allocated.
==32057==
==32057== searching for pointers to 9,475 not-freed blocks.
==32057== checked 1,545,680 bytes.

                                    (several lines suppressed here)

==32057== 400,040 bytes in 1 blocks are still reachable in loss record 4 of 5
==32057==    at 0x4A05809: malloc (vg_replace_malloc.c:149)
==32057==    by 0x4C798FD: add_heap (gc.c:383)
==32057==    by 0x4C6E915: ruby_init (eval.c:1388)
==32057==    by 0x40AA54: ruby_boot() (ruby_helper.cc:90)
==32057==    by 0x40508A: experiment_list::experiment_list() (experiment_list.cc:44)
==32057==    by 0x4040D7: main (lists.cc:66)
==32057==
==32057==
==32057== 411,214 bytes in 9,310 blocks are still reachable in loss record 5 of 5
==32057==    at 0x4A05809: malloc (vg_replace_malloc.c:149)
==32057==    by 0x4C7B156: ruby_xmalloc (gc.c:144)
==32057==    by 0x4CC1C8B: st_init_table_with_size (st.c:159)
==32057==    by 0x4C986CD: Init_sym (parse.y:5998)
==32057==    by 0x4C7F0E8: rb_call_inits (inits.c:53)
==32057==    by 0x4C6EA16: ruby_init (eval.c:1398)
==32057==    by 0x40AA54: ruby_boot() (ruby_helper.cc:90)
==32057==    by 0x40508A: experiment_list::experiment_list() (experiment_list.cc:44)
==32057==    by 0x4040D7: main (lists.cc:66)
==32057==
==32057== LEAK SUMMARY:
==32057==    definitely lost: 0 bytes in 0 blocks.
==32057==      possibly lost: 0 bytes in 0 blocks.
==32057==    still reachable: 816,531 bytes in 9,475 blocks.
==32057==         suppressed: 0 bytes in 0 blocks.

                                    (several lines suppressed here)
```

This type of leak could signal a potential flaw in the system libraries, that might, or not, be retaining allocated memory just in case another application that uses the library might need it.

As the example shows, apart from a memory leak summary, Valgrind actually logs the call stack for every type of memory leak, so that the process of debugging the offending functions is greatly simplified. In the particular example shown above, the eventual leaks

are produced around some Ruby related operations, hinting that the interaction between C + + and Ruby objects has to be checked.

The following example (corresponding to the `Ruby_test` tool for MatCalc) shows all classes of leaks:

```
==32139== My PID = 32139, parent PID = 29514.  Prog and args are:
==32139==    ./Ruby_test
--32139-- Command line
--32139--    ./Ruby_test
--32139-- Startup, with flags:
--32139--    -v
--32139--    --tool=memcheck
--32139--    --leak-check=yes
--32139--    --show-reachable=yes
--32139--    --log-file=/u0/grad/sancheg/Out/valgrind.log
==32139==
==32139== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 4 from 1)
==32139== malloc/free: in use at exit: 694,548 bytes in 8,123 blocks.
==32139== malloc/free: 8,450 allocs, 327 frees, 704,697 bytes allocated.
==32139== searching for pointers to 8,123 not-freed blocks.
==32139== checked 1,412,776 bytes.
==32139==
==32139== 8 bytes in 1 blocks are definitely lost in loss record 1 of 8
==32139==    at 0x4A06019: operator new(unsigned long) (vg_replace_malloc.c:167)
==32139==    by 0x403693: ruby_test_experiment_runner() (Ruby_test.cc:150)
==32139==    by 0x403D23: main (Ruby_test.cc:210)
==32139==
==32139== 8 bytes in 1 blocks are still reachable in loss record 2 of 8
==32139==    at 0x4A06019: operator new(unsigned long) (vg_replace_malloc.c:167)
==32139==    by 0x412BDF: ruby_new(char const*, int, ...) (ruby_helper.cc:168)
==32139==    by 0x403B71: ruby_test_ruby_call_method() (Ruby_test.cc:94)
==32139==    by 0x403CFC: main (Ruby_test.cc:206)
==32139==
==32139== 32 bytes in 1 blocks are possibly lost in loss record 3 of 8
==32139==    at 0x4A05809: malloc (vg_replace_malloc.c:149)
==32139==    by 0x4C7B156: ruby_xmalloc (gc.c:144)
==32139==    by 0x4C97C2E: top_local_setup (parse.y:5801)
==32139==    by 0x4C9F393: ruby_yyparse (parse.y:369)
==32139==    by 0x4CA886F: yycompile (parse.y:2692)
==32139==    by 0x4CBE46C: load_file (ruby.c:974)
==32139==    by 0x403C87: ruby_basic_test() (Ruby_test.cc:58)
==32139==    by 0x403CD5: main (Ruby_test.cc:202)

                                (several lines suppressed here)

==32139== 400,040 bytes in 1 blocks are still reachable in loss record 8 of 8
==32139==    at 0x4A05809: malloc (vg_replace_malloc.c:149)
==32139==    by 0x4C798FD: add_heap (gc.c:383)
==32139==    by 0x4C6E915: ruby_init (eval.c:1388)
==32139==    by 0x403BF5: ruby_basic_test() (Ruby_test.cc:40)
==32139==    by 0x403CD5: main (Ruby_test.cc:202)
==32139==
==32139== LEAK SUMMARY:
==32139==    definitely lost: 392 bytes in 3 blocks.
==32139==      possibly lost: 32 bytes in 1 blocks.
==32139==    still reachable: 694,124 bytes in 8,119 blocks.
==32139==         suppressed: 0 bytes in 0 blocks.
```

The combined results of the two latter examples would indicate that the leak is probably produced in the application code, rather than around system libraries (as expected).

As commented in section 5.3.6.4 (p. 85) the development of the Material Model Parameters Fitting Tool for MatCalc unveiled a severe memory leak underlying the core library modules, which at the time of the original development (up to September 2007) had gone unnoticed.

The reason for the problem not being noticed in the first place resides in the combination of two facts: the first, is that Virtlab library development was mainly performed using *ad-hoc* command-line executables for different but fixed problems, which were run just once; the second fact, is that a *single* run of the FEM engine does not generate any directly observable leak – only when the engine is run *repeatedly*, and without actually quitting the application, is the leak actually produced and only then can it be directly observed as a progressive memory shortage (as reported by system tools like the `top` command).

In the aforementioned context, the use of the GUI frontend – which allows repeated simulations without actually exiting the program – will effectively produce the leak; yet, the total leak caused by a few iterations can still go unnoticed in modern computers, fitted with significant amounts of RAM memory as compared to the memory grabbed in each iteration[70]. However, the *hundreds* of iterations required by the parameter fitting tool can actually bring any machine down to its knees, as shown in Fig. 39, where the free swap space of the server `mills.mcmaster.ca` was left exactly in 0KB after the offending tool took over 97.9% of the system's memory[71]:

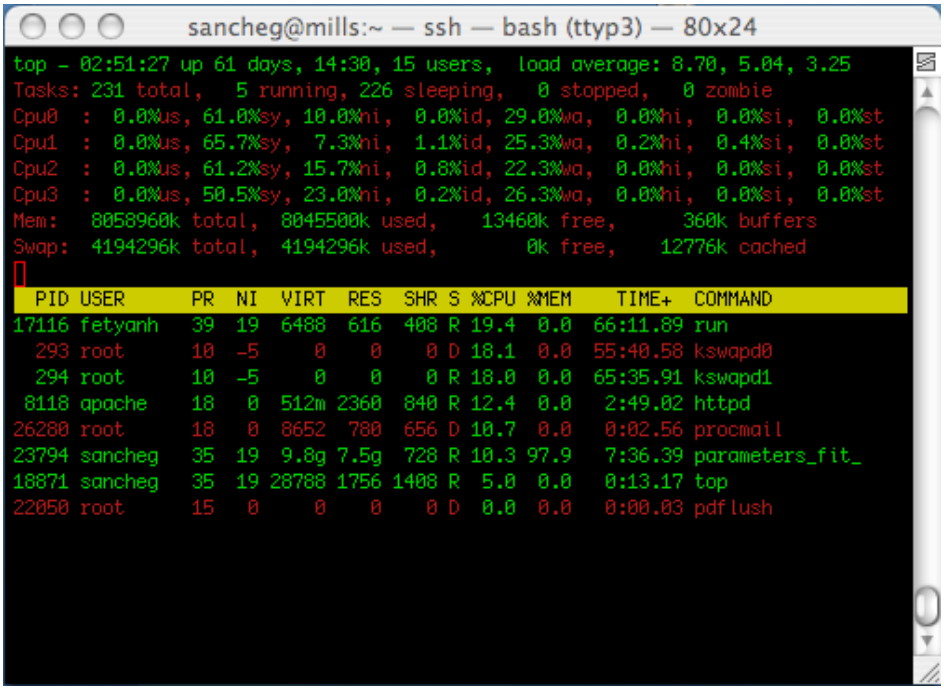


Fig. 39: Virtlab v. 0.1 library memory leak, 'flooding' 97.9% of the computer's memory.

[70] Which for the sample test case used throughout, amounts to roughly 10MB – the size of the `dataset` structure that holds the complete simulation results.

[71] Leaving the system *unusable* for about 15 minutes (wall-clock time); it is assumed that a system's garbage collector could correct the situation. The memory 'flooding' was 'tested' three times, with identical results.

The problem was tracked by the use of Valgrind memory checker module, and the most severe leak was almost immediately isolated and fixed by the original developer upon observing Valgrind's logs. The leak and the corresponding fix are described extensively in section 9.13, starting at p. 194.

> This is a clear example of how adequate software engineering practices and appropriate tools can significantly improve the performance and reliability qualities of any software package.

It needs to be stressed that the use of Valgrind in Virtlab v. 1.1 *still* shows the occurrence of memory leaks, that therefore need to be resolved. Yet, their practical impact is tolerable enough for the software package to be usable whilst the leaks are fixed.

> The use of memory checkers in unit and regression tests will alert of memory leaks in a software package as from their inception. These kind of tools should be a key resource in the development of any software package.

An *ad-hoc* attempt at regression testing for MatCalc was started with the creation of a shell script, `test_matcalc.sh`, that runs a series of memory leak checks using the memory checker tool in Valgrind on various MatCalc v. 1.1 executables: `lists`, `dset_diff`, `configset_test`, `FEM_test`, `Ruby_test`, and `matcalc` itself. For the memory leaks check, just depending on the different exit paths the executable has, at least a check has to be performed to verify if that particular termination path has not produced a leak. The script (listed in section 9.10, p. 184) gives an idea of the various testing combinations the referred executables can undergo for this leak check.

[6.5] System tracing tools

"As a systems administrator, and often as a developer, you are more likely to be interested in finding a fault with a program. This can be why a particular program is causing other problems (such as memory and other errors), or why an application is not behaving as it should, and how it has in the past. Debugging the specifics of the application in this instance are not often useful. Instead, you need to examine how the application is being executed by the operating system.

*With **debugging**, you are examining the execution of the individual functions defined within the application. Debugging concentrates on the application and the functions and structures within it, but typically ignores the system calls and calls to library functions that are made by the application to the operating system. Debugging provides a wealth of information about your application, but not necessarily how the operating system is executing that application.*

*With **tracing**, you are monitoring the interaction between the application and the operating system, usually by looking at the operating system functions that are called by the application during its execution.*

*Despite these semantic differences, **the major difference between tracing and debugging is that tracing can take place without you having to have access to the source code, or to compile the application in any special way. This means that you can trace an application that comes with the operating system, or from a third-party vendor** to find out what is wrong."*[72] ([Bro09], p. 4).

Tracing an application enables you to find out:

- Memory usage and calls to map memory
- Files being opened and closed during execution
- Reads and writes to different files
- Libraries loaded for a given application

Tracing tools, therefore, are usually included among the operating system tools:

- **Linux**: `strace`, `ltrace`
- **Solaris/OpenSolaris**: `truss`; `dtrace`
- **MacOS X**: `ktrace`, `dtrace`, `dtruss`

Typically, a tracing tool is used either to *launch* an application (so its interaction with the operating environment can be traced, and observed) or to *attach* to a currently running application, for the same purposes. For the latter case, a wealth of other system tools such as `ps` (and its 'relatives') allow to examine the running processes in the system and observe/obtain useful information for understanding their behaviour in relation to the operating environment. Similarly, tools like `ldd` in Solaris or Linux list the dynamic dependencies of executable files or shared objects , and can provide additional help to track down issues between the application and the operating environment.

[72] The boldface type applied here if for 'highlighting' sections of particular relevance in the quote; it is *not* part of the original text.

The following example, copied verbatim from [Bro09] (p. 8) assumes the existence of an application, `errnoacc`, which fails with a not altogether informative error message:

```
$ ./errnoacc
ERROR: Application failed to initialize
```

Either in the absence of the source code for the application, or because the issue is suspected to lie in the interaction with the system rather than internally, tracing is done in place of debugging. The following lines are the typical output of the `trace` tool, applied to the malfunctioning program of the example (copied verbatim from [Bro09], p. 9):

```
$ truss ./errnoacc
execve("errnoacc", 0x08047B20, 0x08047B28) argc = 1
mmap(0x00000000, 4096, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, -1, 0)
    = 0xFEFB0000
resolvepath("/usr/lib/ld.so.1", "/lib/ld.so.1", 1023) = 12
getcwd("/export/home/mc", 1014)                     = 0
resolvepath("/export/home/mc/errnoacc", "/export/home/mc/errnoacc", 1023) = 24
xstat(2, "/export/home/mc/errnoacc", 0x080477E4) = 0
open("/var/ld/ld.config", O_RDONLY)                 = 3
fxstat(2, 3, 0x080476C4)                            = 0
mmap(0x00000000, 144, PROT_READ, MAP_SHARED, 3, 0) = 0xFEFA0000
close(3)                                            = 0
sysconfig(_CONFIG_PAGESIZE)                         = 4096
xstat(2, "/usr/lib/libc.so.1", 0x08046F04)          = 0
resolvepath("/usr/lib/libc.so.1", "/lib/libc.so.1", 1023) = 14
open("/usr/lib/libc.so.1", O_RDONLY)                = 3
mmap(0x00010000, 32768, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_ALIGN, 3, 0) = 0xFEF90000
mmap(0x00010000, 1413120, PROT_NONE, MAP_PRIVATE|MAP_NORESERVE|MAP_ANON|MAP_ALIGN,
    -1, 0) = 0xFEE30000
mmap(0xFEE30000, 1302809, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_TEXT, 3, 0)
    = 0xFEE30000
mmap(0xFEF7F000, 30862, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
    MAP_INITDATA, 3, 1306624) = 0xFEF7F000
mmap(0xFEF87000, 4776, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_ANON,
    -1, 0) = 0xFEF87000
munmap(0xFEF6F000, 65536)                           = 0
memcntl(0xFEE30000, 187632, MC_ADVISE, MADV_WILLNEED, 0, 0) = 0
close(3)                                            = 0
mmap(0x00010000, 24576, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON|MAP_ALIGN,
    -1, 0) = 0xFEE20000
munmap(0xFEF90000, 32768)                           = 0
getcontext(0x08047534)
getrlimit(RLIMIT_STACK, 0x0804752C)                 = 0
getpid()                                            = 15727 [15726]
lwp_private(0, 1, 0xFEE22A00)                       = 0x000001C3
setustack(0xFEE22A60)
sysi86(SI86FPSTART, 0xFEF879BC, 0x0000133F, 0x00001F80) = 0x00000001
open("/etc/shadow", O_RDONLY)                       Err#13 EACCES [file_dac_read]
ioctl(1, TCGETA, 0x08046BB0)                        = 0
fstat64(1, 0x08046B10)                              = 0
ERROR: Application failed to initialize
write(1, " E R R O R :     A p p l i".., 40)       = 40
_exit(0)
```

From the trace, it can be seen that the issue is in this 6th line counting from the bottom, that reads:

`open("/etc/shadow", O_RDONLY) Err#13EACCES [file_dac_read].`

The problem with the application is that it is trying to open a file to which the user running the application has no permission to access.

As can be inferred from the aforementioned tracing example, each line of output shows the application executing one function call, alongside with the arguments to the function and its call return value. Unlike in debugging, each function call listed is a function within the system or system libraries, and therefore represents a lower-level interface to the functions called. For example, opening a file within an application may use the `fpopen()` function within C or C++, but this is in fact a wrapper to the more primitive `open()` function ([Bro09], pp. 7-8).

The tracing tool `strace` was actually used by the System Administrator in CAS Department at McMaster University, for troubleshooting an initialization problem of the Maple computer algebra system (installed in the CAS server `mills.cas.mcmaster.ca`) from within MatGen code generator for Virtlab. The proper use of the tool showed the application, MatGen, failing to find some Maple shared objects (by searching for them underneath its working directory, as opposed to the system locations):

```
21842 ...
21842
21842 open("/proc/stat", O_RDONLY)      = 8
21842 fstat(8, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
21842 mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x2aeaa0ba2000
21842 read(8, "cpu  1072422 202217 502349 93253"..., 4096) = 886
21842 read(8, "", 4096)                 = 0
21842 close(8)                          = 0
21842 munmap(0x2aeaa0ba2000, 4096)      = 0
21842
stat("/nfs/u0/grad/sancheg/Projects/virtlab/svn/mapleclient/src/ATHLON641024SSE3_4/libgmp
.so", 0x7fff0ef40910) = -1 ENOENT (No such file or directory)
21842
stat("/nfs/u0/grad/sancheg/Projects/virtlab/svn/mapleclient/src/ATHLON641024SSE2_4/libgmp
.so", 0x7fff0ef40910) = -1 ENOENT (No such file or directory)
21842
stat("/nfs/u0/grad/sancheg/Projects/virtlab/svn/mapleclient/src/ATHLON641024SSE3_2/libgmp
.so", 0x7fff0ef40910) = -1 ENOENT (No such file or directory)
21842
stat("/nfs/u0/grad/sancheg/Projects/virtlab/svn/mapleclient/src/ATHLON641024SSE2_2/libgmp
.so", 0x7fff0ef40910) = -1 ENOENT (No such file or directory)
21842
stat("/nfs/u0/grad/sancheg/Projects/virtlab/svn/mapleclient/src/ATHLON641024SSE3/libgmp.s
o", 0x7fff0ef40910) = -1 ENOENT (No such file or directory)
21842
stat("/nfs/u0/grad/sancheg/Projects/virtlab/svn/mapleclient/src/ATHLON641024SSE2/libgmp.s
o", 0x7fff0ef40910) = -1 ENOENT (No such file or directory)
21842
stat("/nfs/u0/grad/sancheg/Projects/virtlab/svn/mapleclient/src/X8664SSE2/libgmp.so",
0x7fff0ef40910) = -1 ENOENT (No such file or directory)
21842 open("/nfs/u0/grad/sancheg/Projects/virtlab/svn/mapleclient/src/libgmp.so",
O_RDONLY) = -1 ENOENT (No such file or directory)
21842 write(2, "fatal error(-1) loading GMP libr"..., 105) = 105
21842 rt_sigprocmask(SIG_UNBLOCK, [ABRT], NULL, 8) = 0
21842 tgkill(21842, 21842, SIGABRT)     = 0
21842 --- SIGABRT (Aborted) @ 0 (0) ---
21842 +++ killed by SIGABRT +++
```

A soft link created to that library (and other shared objects, also sought in the inappropriate place) allowed the application to initialize.

Portability and performance issues with an application might be worth investigating in a ‘system wide’ fashion, rather than ‘application centric’. Awareness of system tools and utilities, like tracing utilities, can provide a rich insight into the behaviour of an application interacting with the underlying operating environment.

[7] CONCLUSIONS & FUTURE WORK/RESEARCH

This chapter evaluates the effects of the modifications and additions effected in the software codebase during the period May 2009 – March 2010, and presents ideas for future work on the software package, as well as future research ideas based on the Virtlab project.

[7.1] Conclusions

The goals set for the period May 2009-March 2010, stated in section 5.2.3 (p. 42) have been globally achieved:

- **The overall *usability* of the software product has been significantly enhanced.**
  - The GUI frontend to MatCalc is intuitive to use, the information is laid out clearly, all the controls work 'as intuitively expected', and the underlying library functionality is completely accessible.
  - A CLI frontend to MatCalc has been developed, and is as functional as the GUI frontend by way of file input and output.
  - Both frontends are compatible in terms of file input/output, and make the exact same use of the underlying simulation engine.
  - MatCalc's building process is fully automated.
  - MatGen's building process, albeit not completely automated, is now appropriately documented.
- **Virtlab library *portability* has been enhanced.**
  - Virtlab now builds on Solaris/OpenSolaris platforms, apart from Linux and MacOS X (the original development platforms).
- **Virtlab library documentation, albeit not complete, has been significantly enhanced, and the future documentation roadmap has been traced thereupon, improving the *understandability* and *maintainability* of the software package.**
  - A systematic and automated documentation of the codebase has been initiated, upon the framework provided by Doxygen.
  - Virtlab's documentation has been enhanced, by way of improving or creating the files listed in chapter 9 (p. 145) as well as specific figures and sections included in this report which can later be included in the appropriate documents (listed in Table 1, p. 15).
- **A Material Model Parameters Fitting tool has been developed to address the *accuracy*, *reliability*, and *verifiability* qualities of the software product.**
  - The tool, currently operational as a serial computing application, is undergoing work towards a parallel version by incorporation of MPI constructs[73].

[73] The parallelization of the tool will not only address time savings in the computation, but will also be used to experiment with slight variations of the underlying algorithm, that exploit the potential advantages that a parallel operation can bring for 'parameter space exploration'.

Additional consequences of the work effected in the period May 2009-March 2010:

- The separation of concerns within the software library has been improved with the addition of file input, file output, and configuration modules, each of which apply information hiding policies to the best extent possible, thus improving the maintainability, understandability and reusability qualities of the software package.
- A severe memory leak residing in MatCalc's modules was detected, diagnosed, and fixed, thus improving the performance quality of the software, and contributing to its reliability.
- Other memory leaks have been uncovered and documented for future resolution, contributing thus to the maintainability of the software, and paving the road to improve its performance and reliability once the problems are fixed.
- The automation incorporated into the code documentation and software package testing, albeit incipient, sets a roadmap for future improvement on both aspects, contributing to the maintainability quality of the software package.
- The benefits provided by the judicious application of software quality concerns in the development of the *original* Virtlab library were constated, and exploited. Evidence of this claim is supported by these facts:
  - The actual ability to refactor existing tools (like the currently named `diff_dset`, `FEM_test`, `lists`, `Ruby_test`) as well as the rapid development of the parameter fitting tool for MatCalc, just by intuitively reusing methods from the existing modules, and easily integrating them with those of newly developed modules.
  - The ability to port the software library to other Unix platforms, and to automate the build process for MatCalc, which – despite the work involved – was feasible under the originally selected building environment (the GNU Build System) and done squarely within its capabilities.

Conclusions that can be drawn from the work effected in the period May 2009-March 2010:

Software Engineering techniques and tools are a necessity in the development of scientific software: as in any kind of complex software system, the need for source version control, task automation – particularly, automated builds and tests – and documentation aids, cannot be understated.

Proper documentation is more important than the code itself: relying solely on the code without documentation is short of the 'Bourbakization' of mathematics[74] – the risk of losing the ideas behind the implementations is unwise and potentially unaffordable.

Scientific software requires, first and foremost, a solid software engineering foundation to comply with its intended qualities of *Reliability*, *Accuracy*, *Performance*, *Understandability*, *Maintainability*, *Verifiability*, *Reusability*, and *Portability*. It only makes sense to delve into scientific computation aspects of the software after there is confidence in the software engineering aspects of the application, and this encompasses not only the application itself, but also its environment – compiler, assembler, linker, loader, operating system libraries, operating environment, and even, the underlying hardware platform.

Overall contribution to the Virtlab Project by way of the work effected in May 2009-March 2010:

Virtlab v. 1.1 is left in a state that enables practical use, and with the indispensable documentation for maintenance and future development.

[74] "*Nicolas Bourbaki is the collective pseudonym under which a group of (mainly French) 20th-century mathematicians wrote a series of books presenting an exposition of modern advanced mathematics, beginning in 1935. With the goal of founding all of mathematics on set theory, the group strove for rigour and generality*. (...) *The emphasis on rigour may be seen as a reaction to the work of Henri Poincaré, who stressed the importance of free-flowing mathematical intuition, at a cost of completeness in presentation*. (...) *It provoked some hostility, too, mostly on the side of classical analysts; they approved of rigour but not of high abstraction*." From: http://en.wikipedia.org/wiki/Bourbaki. G. Chaitin (http://www.cs.auckland.ac.nz/~chaitin/) has criticized Bourbaki's group policy of polishing mathematical proofs 'until perfection', on the grounds that the ideas behind the proof are progressively lost on that quest for attaining most concise formal proof possible; in this sense, much like keeping the latest revision of the code but devoid of the documentation that explains what it is intended to do, and the design decisions thereafter.

[7.2] Future work

Having reached a minimal 'usability' grade, future work on the Virtual Material Laboratory Project software library will pursue the following objectives[75] to leverage it to 'industrial' or 'production' grade:

- **Fix the code generation problem of MatGen.**
- **Improve the documentation**:
  - produce the complete set of documents listed in Table 1 (p. 15): the **VL_TM** document should actually be a *document set* − **System Requirements Specification (SRS)**; **Module Guide (MG)**: **Module Interface Specification (MIS)** − as suggested in [HS99].
  - complete the source code documentation 'brief" and 'detailed' comments, for use by the Doxygen documentation tool.
  - document the objects (classes, methods, functions, structures, etc.) in the codebase using Doxygen's appropriate keywords, to improve the generated documentation thereafter.
  - try to automate the maintenance of the complete documentation set as much as possible.
- **Implement proper unit and regression testing**, linked to the `make check` target of the build system.
- **Fix the documented memory leaks.**
- **Improve the building process** to achieve:
  - support for running unit and regression tests at the building stage, with `make check`.
  - elimination of any 'hardcoded' paths, leaving `make install` target operational.
  - complete automation of MatGen building process.
  - support for `--without-gui` option to enable building the software package.
  - support for other `--with-<feature>` or `--without-<feature>` options, as needed.
- **Improve the GUI frontend to MatCalc** to achieve:
  - a 'Stop' button to interrupt the simulation.
  - a 'progress bar' or 'percentage indicator' to provide feedback on the simulation running status.
  - better performance and capabilities of the graphics panel and/or capabilities to link to an external plotting utility and leave the current graphics panel as a

[75] These include a prioritized summary of sections 5.3.1.3, 5.3.2.3, 5.3.4.2, 5.3.5.6, 5.3.6.5, 5.3.7.6, and 5.3.8.5.

'fallback' option (this requires support in the building process, by way of appropriate `--with-<feature>` or `--without-<feature>` options).

- **Improve the separation of concerns** to achieve:
  - 'memory safety' in all material test and material model operations.
  - the choice of using the BLAS libraries (http://www.netlib.org/blas/) in lieu of the currently implemented classes for matrix/vector operations (this requires support in the building process, by way of appropriate `--with-<feature>` or `--without-<feature>` options).
- **Improve the FEM algorithm at the core of MatCalc**, either by optimization of the existing code, or substitution of the currently implemented algorithm.
- **Improve the performance of the Material Model Parameters Fitting Tool**:
  - apply MPI constructs for parallel computation (requires redefining convergence criteria)

[7.3] Future research

Future research envisioned around Virtlab project could include:

- Research into automated documentation tools and techniques towards the purpose of 'faking' the rational process promoted in [PC86], and facilitating the maintenance of *up-to-date and consistent documentation.*
- Research of *automatic differentiation techniques* to substitute the symbolic computation of the required derivatives for the material classes formulations, and the automated code generation stage that follows the symbolic manipulation, and comparison of the numerical and symbolic/generative approaches thereafter.

[8] REFERENCES

[Bro09] "*Solutions for tracing UNIX applications*"; M. Brown, 31 Mar 2009. IBM developerWorks® (ibm.com/developerWorks) © Copyright IBM Corporation 2009.

[Col00] *Subversion (SVN): version-control system*; CollabNet, Elego, WANdisco; initiated in 2000 by CollabNet Inc. URL: http://subversion.apache.org

[CSMAK07] "*Model Manipulation as Part of a Better Development Process for Scientific Computing Code*"; J. Carette, S. Smith, J. McCutchan, C. Anand, A. Korobkine; SQRL Report 48, McMaster University. December 2007.

[Eat94] *GNU Octave: a high-level language, primarily intended for numerical computations*; J. Eaton *et al*, Department of Chemical Engineering , University of Wisconsin, 1994. Copyright © 1998-2006 John W. Eaton. URL: http://www.gnu.org/software/octave

[GJ03] "*Learning the GNU development tools*", edition 0.11.4; E. Gkioulekas (Department of Applied Mathematics - University of Washington), M. Jimenez (Pontificia Universidade Católica do Rio de Janeiro, PUC-Rio). Last updated: 3 April 2003. Copyright © 1998-2003 Eleftherios Gkioulekas and Marcelo Roberto Jimenez. All rights reserved.

[Gla98] *Glade Interface Designer: cross-platform graphical user interface builder for GTK+*; GNOME Foundation, 1998. URL: http://glade.gnome.org

[Gro96] *Message Passing Interface: language-independent communications protocol used to program parallel computers*; W. Gropp *et al*, 1996. OpenMPI is an Message Passing Interface (MPI) library project combining technologies and resources from several other projects (FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI). URL: http://www.open-mpi.org

[HS99] "*Software Design, Automated Testing, and Manteinance - A Practical Approach*" (electronic edition); D. Hoffman, P. Strooper. July 22, 1999.

[KP97] *Gtk+: GNU Image Manipulation Program GIMP Toolkit*; S. Kimball, P. Mattis, eXperimental Computing Facility (XCF) at University of California, Berkeley, 1997. Cross-platform widget toolkit for creating graphical user interfaces. Maintained by the GNOME Foundation. URL: http://www.gtk.org

[Map80] *Maple: general-purpose computer algebra system*; Symbolic Computation Group, University of Waterloo, Waterloo, ON (Canada). Since 1988, developed and sold commercially by Waterloo Maple Inc. (a.k.a. Maplesoft) Waterloo, ON (Canada). URL: http://www.maplesoft.com

[Mat95] *Ruby: dynamic, reflective, general purpose object-oriented programming language*; Y. Matsumoto *et. al.*, 1st release: December 21, 1995. URL: http://www.ruby-lang.org/en

[McC07] "*A generative approach to a virtual material testing laboratory*"; J. McCutchan; M. Sc. Thesis; McMaster University. September 2007.

[PC86] "*A Rational Design Process: How and Why to Fake It*"; D. Parnas, P. Clements, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, Feb. 1986, pp. 251-257. 0098-5589/86/0200-0251$01.00 Copyright © 1986 IEEE.

[San09] "*CES*701 Project Proposal*"; G. Sánchez; School of Computational Engineering and Science, McMaster University. Project proposal for CES*701 'Foundations of Modern Scientific Programming' course, 2009 (unpublished).

[San10] "*CES#713 Project Proposal*"; G. Sánchez; School of Computational Engineering and Science, McMaster University. Project proposal for CES#713 'The Message Passing Interface for Parallel Applications' course, 2010 (unpublished).

[Sew00] *Valgrind: programming tool for memory debugging, memory leak detection, and profiling*. J. Seward *et. al*. Copyright © 2000-2009 (Authors).URL: http://valgrind.org

[SHL83] "*Civil Engineering Systems Analysis and Design*"; A. Smith, E. Hinton, R. Lewis; Copyright © 1983 John Wiley and Sons Ltd. ISBN 0-471-90060-5.

[SMC07] "*Program Families in Scientific Computing*"; S. Smith, J. McCutchan, F. Cao; Computing and Software Department, McMaster University. 7th. OOPSLA Workshop on Domain Specific Modelling (DSM'07), pp. 37-49.

[SMC09] "*Commonality Analysis for a Family of Material Models*"; S. Smith, J. McCutchan, J. Carette; Computing and Software Department, McMaster University. Draft Technical Report (unpublished). May 6, 2009.

[SY09] "*A document driven methodology for developing a high quality Parallel Mesh Generation Toolbox*"; S. Smith, W. Yu; Computing and software Department, McMaster University. Copyright © 2009 Elsevier Ltd. Available: June 2009, 'Advances in Engineering Software' Journal. doi:10.1016/j.advengsoft.2009.05.003

[VETT00] *"GNU Autoconf, Automake and Libtool";* G. Vaughan, B. Elliston, T. Tromey, I. Taylor. Copyright © 2000, 2006 Gary V. Vaughan. URL: http://sources.redhat.com/autobook.

[vHe97] "*Doxygen - Source code documentation generator tool*"; D. van Heesch. Copyright © 1997-2009 by Dimitri van Heesch. URL: http://www.stack.nl/~dimitri/doxygen/ (redirected from http://www.doxygen.org).

[WK86] *Gnuplot: portable command-line driven graphing utility for linux, OS/2, MS Windows, OSX, VMS, and other platforms*; T. Williams, C. Kelley. Copyright © 1986-1993, 1998, 2004 Thomas Williams, Colin Kelley. URL: http://www.gnuplot.info

[9] APPENDIX

This chapter lists the contents of specific documentation files for the Virtlab software package, and is meant as a reference while the documents listed in Table 1 − where the information herein should be appropriately included − undergo development.

- Sections 9.1, 9.2, 9.3 and 9.4 contain the copy of the `ChangeLog` and `TODO` files for MatCalc and MatGen, respectively. These are copied 'as is', and therefore have not been revised neither for informal/offending language, nor spelling mistakes, and the like issues. They are intended as an example of the effort/difficulty involved in maintaining the software package as it evolved. Additionally to these records, an effort to maintain the Subversion log file was effected, but the duplication of tasks will probably − almost inevitably − imply the existence of both 'sins of omissions and commissions' between the `ChangeLog` files and corresponding Subversion log entries. The latter are *not* listed herein.
- Section 9.5 lists the complete contents of the file `INSTALL2`, which provides information on the specific issues of compile time and run time linking.
- Section 9.6 lists the contents of the file `matcalc_driver.txt`, which provides a succinct example of how to use Virtlab's codebase as a library for simulation.
- Section 9.7 lists the contents of the file `add_experiment.txt`, which explains how to define a virtual material test for use in MatCalc.
- Section 9.8 lists the contents of the file `add_material.txt`, which details the procedure for adding a material model class to MatCalc.
- Section 9.9 lists the contents of the file `generate_material.txt`, which details the procedure for generating a material model class using MatGen.
- Section 9.10 lists the contents of the shell script `test_matcalc.sh`, which performs a number of memory leak checks using Valgrind (see section 6.4, p. 123) on several of MatCalc tools.
- Section 9.11 lists the contents of `test_EvalMapleStatement.cc` C++ source code file, which uses the `EvalMapleStatement()` function defined in Maple's application programming interface, to execute Maple commands and get their result.
- Section 9.12 contains a copy of the source code file listing for MatCalc v. 1.1, generated by Doxygen.
- Section 9.13 reports the memory leak problem in MatCalc v. 0.1 and its fix[76].
- Section 9.14 presents an overview of the preprocessing, compilation, assembly and static/dynamic linking process.

[76] (Note: the content of this section is, in its entirety, based on the information provided by original developer of Virtlab, John McCutchan, who diagnosed the problem and proposed the corresponding fix).

[9.1] MatCalc `ChangeLog`

```
================================================================================
2009/02/10  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * gui.cc           Fixed experiment, material and algorithm names string
                    handling in function on_open(), which made Ruby abort at
                    runtime on mills.mcmaster.ca; curiously, the bug was not
                    fatal on other platforms.

================================================================================
2009/02/06  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * Doxyfile         MatCalc 1.1, 1.0 and 0.2 Doxyfiles updated to ignore .svn/
                    folders, and to avoid generating man/ documentation.

 * build_matcalc.sh Build scripts updated, and changes propagated to prior
                    versions 1.0 and 0.2.

 * doc_matcalc.sh   Doxygen documentation generation script created, that
                    produces a .tar file under the doc/ folder, containing the
                    html files; the .tar file unpacks to doc/html folder,
                    but this is now ignored by SVN, so the user must unpack the
                    files 'by hand'. Script was propagated to prior versions
                    1.0 and 0.2.
                    NOTE: all html documentation was deleted, and .tar files
                          placed in lieu of it; SVN was mishandling the big
                          number of generated files, with very long filenames.

 * test_matcalc.sh  Test script created to run several executables (using all
                    command-line options where applicable) through Valgrind, to
                    check for memory leaks.
                    NOTE1: Valgrind must be installed and in the user's PATH to
                           operate.
                    NOTE2: The script runs Valgrind with 'full' output, so that
                           if is run overnight will produce complete reports for
                           each case tested, and would avoid the developer to
                           manually re-run a test if something is found to go
                           wrong. Given this, the script then uses grep to try
                           to produce a summary of the results, but this filter
                           still requires a second or third pass to produce a
                           better report.


================================================================================
2009/02/04  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * configset.cc     Reworked to avoid potential leaks due to char* handling.

 * output.cc        Eliminated 'static' property on experiment & material lists
                    (probably inherited from previous versions of the codebase,
                    particularly the gui.cc original file) which caused the
                    actual version of gui.cc to crash in its second attempt to
                    write a file.

 * cli.cc           Error condition handling/messages have been improved.

 * gui.cc           Error condition handling/messages have been improved.
                    The graph panel now can show the shear stress/strain
                    (directions XY, XZ, and YZ).

 * data/matcalc.glade   The graph panel now can show the shear stress/strain
                        (directions XY, XZ, and YZ) matching changes in gui.cc.
                        Default edit boxes values for specimen dimensions were
                        set to 1.000E+00.
```

```
 * FEM_test.cc      More potential leaks fixed. Resource freeing is now done
                    through a function, taking advantage of the 'monolithic'
                    sort of executable.

================================================================================
2009/02/03  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * configset.cc     Fixed malloc(strlen()) calls to malloc(strlen()+1); strlen()
                    does NOT include the terminating NULL character, and
                    Valgrind logs showed invalid writes/reads 1 byte beyond
                    allocated space.

 * FEM_test.cc      String handling now uses (char*)malloc(strlen()+1) dynamical
                    memory allocation, and corresponding free() calls for
                    deallocation. Valgrind logs don't complain of leaks or
                    invalid R/W operations on strings anymore.

 * gui.cc           String handling through gchar* and g_free() revised.

 * ruby_helper.cc   Call to free(argv) added to ruby_new() method.

 * Ruby_test.cc     Test messages improved for information and readability.

 * dset_diff.cc     Tool expanded to compute rel error on shear stresses.

 * experiments/*.rb Changed all experiment files to return TRUE for the gemoetry
                    update (thus, all experiments are returning the true stress,
                    not the enginering one). Deleted all useless ';' therein.

 * experiment_runner.cc Moved again the dataset_ruby::destroy() methods to the
                        last possible position in each method that uses them.

 * matcalc_driver.cc    Changed 'Lists' to 'lists' in messages.

================================================================================
2010/01-02  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * ruby_helper.h        Fixed huge memory leak derived from the interaction
   datachunk_ruby.h     between Ruby and C++ (diagnostic & fix: John McCutchan):
   m_vector_ruby.h      - Data_Make_Struct macro now passed a pointer to free()
   dataset_ruby.h       - datachunk_ruby::create(), dataset_ruby::create(), and
   experiment_runner.cc   m_vector_ruby::create() now call the construct()
                          operator instead of using the assignment one, and
                          corresponding destructor functions were created to
                          match the constructor calls.
                          NOTE: Details of the above are provided in a specific
                                document for the issue.

 * ruby_helper.cc       Fixed leaks generated by the configset module, and
   configset.cc         fixed leak in the ruby_call_method() (diagnostic & fix:
   configset.h          JohnMcCutchan).
   cli.cc                   - new_cs(), del_cs(), set_<member>(), get_<member>()
   gui.cc                     functions implemented in configset modules, and
   input.cc                   all dependencies updated to make use of these;
   matcalc.cc                 Valgrind memory checker tool reports leaks are
   matcalc_driver.cc          now null.
   output.cc                - strdup() function was eliminated in all files for
   test/configset_test.cc     being considered as potentially non-portable.
   test/FEM_test.cc         - strcpy() functions are used instead of the safer
   tools/parameters_fit/\     strncpy() functions, since all string handling is
     parameters_fit_tool.cc   done through char* to dynamically allocated memory
     (now moved to file:      of the appropriate size.
     tools/hooke_jeeves.cc)

 * test/FEM_test.cc     Modifications to FEM_test.cc try to fix memory leaks
```

```
                        therein, yet code still needs revision.
                        NOTE: FEM_test specifically 'hardcodes' many functions
                              available in e.g. input/output modules on purpose,
                              to be independent of those, and just provide the
                              most 'direct' way to test the FEM engine itself.
                              Caveats of this approach are that the programmer
                              is responsible to duplicate any changes done in
                              the core codebase to this specific test.

* configset.cc          Function prn_cfg() added for dumping the defaults;
  configset.h           the function is only visible to the configset module,
                        but could be made public as a debug aid, for printing
                        the current configuration whenever it is invoked.

* tools/hooke_jeeves.cc (See entry for tools/parameters_fit/
                        parameters_fit_tool.cc)

* tools/parameters_fit/parameters_fit_tool.cc
                        Moved to tools/hooke_jeeves.cc.

* tools/lists.cc        (See entry for test/Lists.cc)

* tools/dset_diff.cc    (See entry for test/Diff_tool.cc)

* test/Lists.cc         Moved to tools/lists.cc.

* test/Diff_tool.cc     Moved to tools/dset_diff.cc; fixed memory leaks, and
                        expanded to compute differences in stress/strain for all
                        specimen's directions (X, Y, Z) between data in two
                        datasets. Can be further expanded to consider the cross-
                        products of stress/strain (directions XY, YZ, XZ).

* test/SM_tool.cc       Eliminated; original purpose seemingly no different than
                        that of Diff_tool.cc, which was improved and moved into
                        tools/dset_diff.

* Makefile.am           Makefile.am, SVN ignore list, and Doxygen documentation
  svn-ignore.lst        updated to current revision.
  (Doxygen documents)


================================================================================
2009/12/09  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * parameters_fit_tool.cc: Alpha version of the material models parameters
                           fitting tool, under:

                      * 1.1/matcalc/src/tools/parameters_fit/

                      - SVN repository has been modified thus, adding 1.1/.


================================================================================
2009/10/16  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * >>> NOTE <<<     This ChangeLog has not been updated for two months, and has
                    not recorded some significant changes during this period.

                    Basically, the major changes since last update are:

                    ____________________________________
                    - SVN repository has been modified:

                      Ironically, against the basic purpose of a VC system, the
                      repository has been divided 'by hand' into 3 branches:
```

```
  * 0.1/  holds the original code from J.McCutchan;

  * 0.2/  holds the code updates done by G.Sanchez, which:
          - improved the build process based on the GNU
            autotools suite (MatCalc build is fully
            automated, and INSTALL documentation updated)
          - allow building for Solaris/OpenSolaris OS's;
          - completed the unfinished features of 0.1/, such
            as enabling file I/O from the GUI, and improving
            warning/error messages handling
          - refactored the GUI to a 'user friendly' state,
            extending the functionality controllable from it
            (such as enabling to control the time step of
            the FEM algorithm and test specimen dimensions);
          - generated basic Doxygen documentation for the
            codebase, including call/caller graphs and
            dependency graphs.

  * 1.0/  holds the code refactored by G.Sanchez from the
          code hosted in 0.2/, which basically addresses
          important separation of concerns:
          - introduces a CLI module to enable 'batch'
            mode simulations (useful for regression testing,
            and for model fitting algorithms 'external' to
            MatCalc, which can be run from shell scripts);
          - introduces a configuration module to hold the
            simulation configuration parameters;
          - moves the file I/O handling from the GUI module
            to dedicated input and output modules;
          - moves the simulation code from the GUI module to
            a specific matcalc 'driver' module

  The rationale for the 'hand made' aforementioned divison
  is that those three major stages in the development of
  MatCalc/MatGen would stand out immediately and clearly,
  avoiding the need to execute SVN commands to surf the SVN
  logs and checkout different revisions, and also avoiding
  the need for future users to determine which revision
  ranges can be conceptually considered as a 'unit'.

  Even though this could have easily been accomplished by a
  proper use of the SVN resources (tags, branches, comments)
  those resources were not properly exploited from the
  beginning, and the task of rewinding the revisions to fix
  them will not be undertaken.

_____________________________________________________
- MatCalc has been significantly improved and extended:

  * MatCalc can run in GUI or CLI mode, from a single
    executable 'matcalc'

  * Separations of concerns has been implemented by the
    creation of dedicated modules:
    - 'matcalc' module that decides which version to run
      (GUI or CLI)
    - 'gui' module that just addresses the GUI issues
    - 'cli' module that just addresses the CLI issues
    - 'input' and 'output' modules that handle file I/O
    - 'configset' module that handles simulation parameters
      configuration for the current run
    - 'matcalc_driver' module that drives the rest of the
      codebase, thus implementing the simulaton engine

  * File I/O handling has been improved and file format
    has ben standardized.
```

```
                    Please read notes for SVN repository above for a more
                    detailed account of these -and other- changes.


================================================================================
2009/08/30  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * FEM_test.cc:     FEM_test (formerly femtest.cc) now accepts CL switches that
                    allow to change timestep, timespan, specimen dimensions, E,
                    Nu, eta, tolerance and initial stress values for a
                    'hardcoded' Uniaxial test for material E2_Lin.Viscoelastic.
                    It now also writes an output file with both the simulation
                    setup and simulation data, in a format that is compatible to
                    the current version of MatCalc GUI.


================================================================================
2009/08/20  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * *.cc:            All informational and error messages now use 'debug_utils.h'
                    defined functions for printing to cout or cerr.
                    All messages include [filename.cc] text to identify the
                    file that produces the message.

 * gui.cc           Write to file now includes header line stating the number of
                    data records on the file.
                    Read from file now warns that lines with NaN's are ignored
                    on read.

                      NOTE: if forced to read a file with NaN's, these are
                      displayed as +0.000E0 on the data table.

================================================================================
2009/08/06  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * gui.cc:          Read method now sets the comboboxes and text entries in
                    MatCalc's GUI, so:
                    - Read from file method is working completely
                    - Write  to file method is working completely


================================================================================
2009/07/28  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * dataset.cc;      Eliminated read/write from/to file methods
   dataset.h:


 * gui.cc:          Added read/write from/to file methods:
                    - Output file contains a header with the simulation setup
                      parameters, and a table with the simulation data
                    - Numerical output is formatted in scientific notation.
                    - NOTE: read method DOES NOT set the comboboxes and text
                            entries in MatCalc's GUI (for the moment, it just
                            prints the simulaton setup to stdout).

 * difftool.cc;     Disabled all references to read/write to/from file in these
   matcalc.cc;      tools, which used dataset.cc methods.
   femtest.cc:      - NOTE: the methods should be enabled once they are cast
                            away from gui.cc and into specific input/output
                            modules.


================================================================================
2009/07/24  Gonzalo Sanchez  <sancheg@mcmaster.ca>
```

```
 * graph_widget.cc: Fixed graph points rendering that yielded an horizontally
                    and vertically 'flipped' graph.


================================================================================
2009/07/24  Gonzalo Sanchez  <sancheg@mcmaster.ca>
 * gui.cc;          Enabled writing/reading simulation results data to/from a
   graph_widget.cc; file (redrawing the graph and data table upon data load).
   dataset.cc:


================================================================================
2009/07/21  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * matcalc.glade:   Modified interface specification file:
                    - Added algorithm combobox

 * gui.cc:          Added support for algorithm selection through combobox


================================================================================
2009/07/21  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * matcalc.glade:   Modified interface specification file:
                    - Changed all labels to reflect units dimensions
                      (e.g. [t], [StressU], [L], [L/L])

 * Sheer.rb:        Corrected spelling:
                    - Changed filename to Shear.rb
                    - Changed Sheer for Shear within applicable the file

 * *integrator.cc:  Added timestep parameter in all files:
                    - Added 'scalar t_step' as parameter to class definitions
                    - Substituted the hardcoded timesteps in each file for
                      the parameter 't_step'

 * gui.cc;          Updated ::integrate() method to include t_step
   graph.cc;
   femtest.cc:


================================================================================
2009/07/20  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * matcalc.glade:   Modified interface specification file:
                    - Deleted tip for graph area pane

 * gui.cc:          Interchanged Stress and Strain axis:
                    - Set the graph X axis for Strain data
                    - Set the graph Y axis for Stress data

                    Modified constants display format:
                    - Changed format "%lf" to "%+#10.3E"

 * graph_widget.cc: Modified format of the data displayed in the graphical pane:
                    - Changed coordinate values from pixels to physical values
                    - Changed displayed string for readability: added 'strain'
                      and 'stress' labels and graph axis limits
                    - Changed format of displayed values to "%+#11.3E"
                    - Changed type of first two members in _MatcalcGraphPrivate
                      from 'int' to 'scalar' to match Cairo's types (uses
                      'doubles' for the cursor coordinates in pixels).
                    - Changed variable names of those pertaining to the grpah
                      display: capitals indicate 'pixel' units, lowercase
                      is for 'physical' units.
```

```
================================================================================
2009/07/14  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * matcalc.glade:   Modified interface specification file:

                    - Added support for resizable graphics/data table panes.
                    - Added edit fields for specimen dimensions and timestep.
                    - Added tips and modified labels to improve usability.

 * gui.cc:          Modified to support changes of interface specification file.


================================================================================
2009/07/03  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * Makefile.am:     Added line to support inclusion of m4 macros included with
                    the software package in the local 'm4/' folder:

                    ACLOCAL_AMFLAGS = -I m4

 * configure.ac:    Added automated check for Ruby (locally distributed m4
                    macro) and ugly hack to adjust linking against 'libruby' at
                    compile- and run- time::

                    AM_CHECK_RUBY(1.8)

                    case "$host" in
                      *-*-mingw*|*-*-cygwin*)
                            if test -e "$ruby_libdir/../../libruby.so" ; then
                              AC_MSG_RESULT([*** found 'libruby.so' at: \
                                             "$ruby_libdir/../../"])
                              RUBY_CFLAGS="-I\$(ruby_archdir)"
                              RUBY_LIBS="-L"$ruby_libdir/../../" \
                                         -Xlinker -rpath -Xlinker \
                                          "$ruby_libdir/../../" \
                                         -lruby"
                              LIBS=" "
                            else
                              AC_MSG_ERROR([*** can't find 'libruby.so' at: \
                                             "$ruby_libdir/../../"])
                            fi
                            ;;
                        *-*-linux*)
                            if test -e "$ruby_libdir/../../libruby.so" ; then
                              AC_MSG_RESULT([*** found 'libruby.so' at: \
                                             "$ruby_libdir/../../"])
                              RUBY_LIBS="-L"$ruby_libdir/../../" \
                                         -Xlinker -rpath -Xlinker \
                                          "$ruby_libdir/../../" \
                                         -lruby"
                              LIBS=" "
                            else
                              AC_MSG_ERROR([*** can't find 'libruby.so' at: \
                                             "$ruby_libdir/../../"])
                            fi
                            ;;
                        *-*-darwin*)
                            if test -e "$ruby_libdir/../../libruby.dylib" ; then
                              AC_MSG_RESULT([*** found 'libruby.dylib' at: \
                                             "$ruby_libdir/../../"])
                              RUBY_CFLAGS="-I\$(ruby_archdir)"
                              RUBY_LIBS="-L"$ruby_libdir/../../" \
                                         -lruby"
                              LIBS=" "
```

```
                          else
                            AC_MSG_ERROR([*** can't find 'libruby.dylib' at: \
                                          "$ruby_libdir/../../"])
                          fi
                          ;;
                      *-*-solaris*)
                          if test -e "$ruby_libdir/../../libruby.so" ; then
                            AC_MSG_RESULT([*** found 'libruby.so' at: \
                                          "$ruby_libdir/../../"])
                            RUBY_CFLAGS="-I\$(ruby_archdir)"
                            RUBY_LIBS="-L"$ruby_libdir/../../" \
                                       -R"$ruby_libdir/../../" \
                                       -lruby"
                            LIBS=" "
                          else
                            AC_MSG_ERROR([*** can't find 'libruby.so' at: \
                                          "$ruby_libdir/../../"])
                          fi
                          ;;
                      *)
                          AC_MSG_WARN([*** Check search and run paths for \
                                       the linker if you have libraries \
                                       installed in non-standard \
                                       locations. Refer to INSTALL2])
                        ;;
                  esac


================================================================================
2009-06-22  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * configure.ac:    Added all macros suggested by 'autoscan' for all supported
                   platforms:

                   AM_PROG_CC_C_O
                   AC_HEADER_STDBOOL
                   AC_HEADER_DIRENT
                   AC_CHECK_FUNCS([pow])
                   AC_CHECK_FUNCS([sqrt])
                   AC_FUNC_STRTOD
                   AC_FUNC_CLOSEDIR_VOID
                   AC_C_INLINE
                   AC_TYPE_SIZE_T
                   AC_TYPE_SIGNAL
                   AC_C_CONST


================================================================================
2009-06-22  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * configure.ac:    Defined the *-*-solaris* platform:

           *-*-solaris*)
               AC_DEFINE(PLATFORM_SOLARIS, 1, [Platform is Solaris/OpenSolaris])
               ;;


================================================================================
2009-06-15  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * src/MCtest.cc:   Modified 'isinf()' and 'isnan()' checks in as suggested in
                   Autoconf manual, sect. 5.5.1, "Portability of C functions",
                   to avoid 'gcc' complaints on Solaris/OpenSolaris platforms:

                   - Eliminated the 'std::' namespace identifier prefix to both
                     'isinf()' and 'isnan()' macros
```

```
                - Added the suggested code:

            /* Autoconf: 5.5.1 Portability of C Functions [G.S. 2009/06/15] */
            #ifndef isnan
            # define isnan(x) \
                (sizeof (x) == sizeof (long double) ? isnan_ld (x) \
                 : sizeof (x) == sizeof (double) ? isnan_d (x) \
                 : isnan_f (x))
            static inline int isnan_f  (float       x) { return x != x; }
            static inline int isnan_d  (double      x) { return x != x; }
            static inline int isnan_ld (long double x) { return x != x; }
            #endif
            #ifndef isinf
            # define isinf(x) \
                (sizeof (x) == sizeof (long double) ? isinf_ld (x) \
                 : sizeof (x) == sizeof (double) ? isinf_d (x) \
                 : isinf_f (x))
            static inline int isinf_f  (float       x) { return isnan (x - x); }
            static inline int isinf_d  (double      x) { return isnan (x - x); }
            static inline int isinf_ld (long double x) { return isnan (x - x); }
            #endif


================================================================================
```

[9.2] MatCalc TODO

```
=== Gonzalo Sanchez ============================================================

[^^] Memory management: check for memory leaks. specially prior to all
     'return()' statements, where possibly char* were forgot to be freed.

     NOTE1: Memory 'flooding' bug solved by John McCutchan; see ChangeLog for an
            overview of the issue and affected files, or the specific document
            on the issue for details.
     NOTE2: 'configset' related leaks are fixed
     WARNING *** Other Ruby related memory leaks persist ***

[  ] Memory management: study memory allocation scheme; it is not necessary to
     keep all data in memory for computation, so it could be dumped to a file
     whenever necessary (especially, when swapping to disk, because it is
     unbearably slow) but maybe for the graphics and data table (GUI version)
     there is no other choice. In the latter case, at least the CLI could be
     provided with a command line switch for the user to choose between
     'full dataset in memory' or 'dataset in file' options.


[  ] Change build scheme so that the material model class files are compiled as
     shared objects, loadable at runtime. That would allow a quicker build of
     MatCalc, which at most would complain of unresolved references if a
     material model class is not present as a .so in the system.

[  ] Define two comboboxes for the graph pane in MatCalc GUI, so each could be
     used to assign an arbitrary COL_DATASET from the dataset to the X or Y
     axis; therefore, the user would be able to 'choose' his/her graph with
     complete flexibility, e.g. against time or against stress.


[  ] Modify the path dependency of the Glade XML interface definition file;
     currently is 'hardcoded' as "../data/matcalc.glade" which then forces the
     execution of 'matcalc' exclusively from its installation folder.


[  ] GUI menus: fix or remove: Copy, Paste, Cut, About; remove others. Note that
     mapleclient's 'About' menu works.


[  ] Improve build options: enable excluding the GUI with switch --without-gui.


[  ] Improve GUI: capture console output on a panel; that would allow the user
     to check the load file errors, the parsed data, and while running the
     simulation, its progress. Alternatively, provide an indication in the
     status bar of the program status, and the simulation progress (i.e. just
     the current line of the console output)

[  ] Remark in documentation that v0.1 GUI (the original version) has the graph
     flipped upside down and sideways, and that as ov v0.2 that has been fixed.

[  ] Study if 'custom' Matrix and Vector classes can be substituted to calls
     to scientific libraries such as BLAS/LAPACK, under the rationale of:
     - reliability (for scientific computation)
     - efficiency
     - maintainability
     - ease of code reuse (for the MatCalc developer, he/she would learn to use
       those libraries for other projects)


[  ] Check if TODO and FIXME issues can be automated directly from the code
```

```
     comments using Doxygen, which would significantly improve the 'low level'
     documentation of the project


[  ] Try to change the algorithms so that they store the initial state (ct=0)
     in the dataset (i.e., that the first datachunk is not that of the first
     step dt) because that will show the initial stresses hardcoded into the
     application....


[  ] Triaxial.rb experiment: uses constants for lx, ly and lz. Check how these
     collide with the settings from the respective text entries from the GUI...
     - can these be simply deleted from here, sinve the integrator takes them
       from the GUI text entry boxes (or CLI switches)?
     - in case that it is not feasible to delete them from the experiment: can
       these be updated from those edit boxes, and therefore, need not appear
       as constants?


[  ] Try to reset the cursor information on the graph pane when 'fill_graph()'
     method is called in 'gui.cc'

[  ] Run 'ifnames' on src code (see Autoconf manual)


[  ] Automated Ruby checks (for all supported platforms)
        [ok] Include 'ruby.m4' macro with software
        [ok] Include AM_CHECK_RUBY(1.8, [...], [...]) in 'configure.ac'file
        [  ] Improve automated Ruby checks in 'configure.ac' file; in particular
             the platform-dependent checks are ugly
        [  ] Wrap the automated Ruby check so that it can be smoothly overriden
             from the 'configure' script command-line with a '--ruby-cflags='
             switch.
             Use the same idea as for the '--disable-debug' switch below:
             ---------------------------------------------------------------------
             # default to debug
             ENABLE_DEBUG=yes
             AC_ARG_ENABLE(debug,
             AC_HELP_STRING([--disable-debug],
                            [disables compilation of debug information]),
             [case "${enableval}" in
                yes) ENABLE_DEBUG=no ;;
                no)  ENABLE_DEBUG=yes ;;
                *)   AC_MSG_ERROR(bad value ${enableval} for --disable-debug) ;;
             esac],
             [ENABLE_DEBUG=yes])
             if test x$ENABLE_DEBUG = xyes; then
                    AC_DEFINE(CACTUS_DEBUG, 1,
                              [Define if debug info should be compiled in])
                    CXXFLAGS="-g"
             fi
             ---------------------------------------------------------------------


[  ] Linker switches for Darwin (MacOS X):
        Check which are the corresponding switches or constructs for the dynamic
        linker 'ld.so' in Darwin, that enable 'hardcoding' into the comiled
        binaries the 'runpath' for dependencies (i.e. which is the equivalent
        switch to '-rpath' or '-R' in Linux and Solaris, respectively).
        Also check how to pass it to the static linker 'ld' from the compiler
        driver's command line (for GNU 'gcc', it is the switch '-Xlinker': see
        INSTALL2 file for an example)


[ok] Memory management: cast to appropriate types and enable all the 'free'ing
     statements for strings in 'gui.cc'.
```

```
[ok] Check if file I/O column width is still compatible with FEM_test.cc; update
     FEM_test.cc in case it is not.


[ok] Fix: reading from file that has NaN's (if not ignoring those lines)
     displays the NaN's as +0.000E0.


[ok] Develop CLI tool with exactly the same characteristics of the GUI tool
     NOTE: see next item.
[ok] Use main() in the gui to accept command line parameters; if none specified,
     or only '-i<input_data> then run the 'full' MatCalc GUI (and in case of an
     input file, display the data). If '--with-gnuplot' specified, use a
     'striped' gui without graph pane (just hide it!) and try to hook to a
     gnuplot window as Octave does. If '-o<ofile>' switch is specified (plus,
     -f<cfg_file>, and maybe, -e<experiment>, -y<yieldf>, -a<alg>, -T<time>,
     -t<tstep> -X<lx> -Y<ly>, -Z<lz> , -E<E>, -N<nu>, -F<constants_file> to
     override the cfg_file contents) then operate in CLI mode.


[ok] Create specific modules for: file input, file output, and configuration
     settings (including comand-line options and configuration files parsing)


[ok] Fix bug: Run after Reset aborts due to constants not set.


[ok] Change formatting of data table values to "%+#11.3E" (probably requires
     editing each cell in a row as text, and apply g_strdup_printf())


[ok] Read method should set the comboboxes and text entries to the values
     specified in the header of the data file. This will require parsing the
     header lines, and setting the comboboxes, text entries, and trees to the
     values specified in each case. Combobox setting will require matching
     strings. The experiment and material parameters will require the use of
     GtkIterator methods.


[ok] Change formatting of output data file values to "%+#11.3E"


[ok] Output the experimental setup alongside the computed data to a text file
     for later reference; either do it in same file (as a header that e.g.
     gnuplot can ignore) or as a separate file, that shares the same root for
     the filename (possibly timestamped, to ensure uniqueness)


[ok] Enable selection of integrator algorithm by the user


[ok] Enable selection of timestep by the user (requires changing the integrator
     algorithms interfaces)


[ok] Check all input and output values default units, and document it (and
     specially in the GUI tool)


[ok] Correct typo error in convergence message: "Didnd't Converge in 5 steps..."


[ok] Check the relationship between the test timing and simulated speed of the
     material model test (viscosity should influence the results if time is
```

```
     changed in a displacement controlled simulation). Some results so far for
     the 'UniaxialX' test with 'vonmises' material are:

        TOTAL_TIME     DISPLACEMENT_X      TIMESTEP        #POINTS
        1              0.1; 0.01; 0.001    0.016666...     59
        10             1; 0.1; 0.01        0.016666...     599

     so it seems that the TIMESTEP is constant, and the software decides the #
     of points to calculate given the TOTAL_TIME; DISPLACEMENT_X seems not to
     intervene in this aspect.


[  ] MatCalc GUI improvement ideas:
-----------------------------------

    [ok] Units:
         -  Add units labels for every edit box in which the user sets values
            or where values are displayed, and to the axis of the graphs.
         -  Change the 'red numbers' that appear in the graphic pane so that
            instead of indicating in pixels both the current cursor's position
            over the graphic's pane and the pane's size, displays the actual
            'physical' values for the plot (is just a scaling between pixels and
            the max value for the two involved columns in the dataset)

            NOTE: the 4 element tuple (Xmouse Ymouse Xmax-1 Ymax-1) of red
            numbers that appear on the graphics pane represent: the actual X Y
            coordinates of the mouse, in pixels, while it hovers the pane (where
            the 0 0 position is the upper left corner, and the Xmax-1 Ymax-1
            position is the lower right corner) and the size Xmax Ymax of the
            graphics pane

    [ok] Help tips:
         -  Add popup help/info tips when hovering with the mouse over the edit
            fields , controls, and output data forms (table & graph) e.g. to
            indicate admissible value ranges for inputs, function of the
            controls, and additional information on the data representation
            such as x,y coordinates on the graph for the other graphical
            projections not displayed

    [  ] Data:
          - Maybe add option for selection of numerical output display
            representation (e.g. scientific, engineering, or standard decimal,
            each with user adjustable field width) both in the output data
            table, and the graphical display?

    [--] Experimental setup:
          - Provide default values that yield 'typical' or 'nice' results (as a
            startpoint for experimenting)
          - Print experiment setup in GUI (nodes constraints) or enable the
            visualization of the script with a button (call an editor to open
            the actual Ruby file, or simply print the file contents in a
            multiline field with scrollbars and fixed width font text).
          - Improve value edition (single click or at most double click to
            select a whole editing field)
          - Consider if the 'Reset' button should reset the last input (think
            this is the best choice, as is now) or the default values
                + maybe add a button 'Stop' that can interrupt an experiment,
                  and in which case leaves the last input values as an aid for
                  slight modifications or typo checking, and make the 'Reset'
                  button effectively reset the experimental setup to the
                  default values after an experiment run has concluded?

    [--] Data table:
          - Avoid superimposing the output data table to the experimental
            parameters, to facilitate checking the graphical results against
            the experimental setup once the experiment is run
```

```
                + maybe table can go in independent window? (check gnuplot idea
                  for graphical output below)
                + maybe the experimental setup should be displayed in the same
                  area that the experiment type selection list and time limit?

          - Improve table display
                + test and improve column autosizing feature
                + maybe add option to fix column width (override autosizing)?
                + maybe add option for selection of numerical output display
                  representation (e.g. scientific, engineering, or standard
                  decimal, each with user adjustable field width?)
    [  ] Graphs:
          - Add units and markings in the graphs axes
          - Add option to 'retain' previous plot so as to compare to it by
            superimposing a new run
          - Add zooming capabilities
          - Change drop list to check boxes, radio buttons, or simple buttons
            to change graph type (but consider that other graphical output
            options might require adding items to each of the alternatives, in
            which case the actual solution is the best, like for the
            experiments)
          - Add option to connect dots with lines in graphs
          - Maybe add option to use gnuplot (if available) as the output data
            graphical display engine, and leave the built-in graphical
            capabilities as a 'fallback' solution? Apart from providing a
            separate screen for the graphical output, which would allow the
            user total flexibility for visualizing the data table, experimental
            setup, and graphical results in acordance to his/her system,
            Gnuplot already has:
                + built in capabilities for axes marking and labeling
                + built in capabilities for  graph title labeling
                + built in capabilities for zooming and cursor tracking
                + built in capabilities for superimposing plots in the same
                  graph, or rendering many graphs 'side by side' in the same
                  window (e.g. could deliver the Stress vs. Strain graphs for
                  all three cooridnates at the same time)
                + is scriptable, and provides a 'reread' command that can track
                  the growth of a data file (which can be tailored to reread
                  from the start of the file, or implement a 'sliding window'
                  over the data)

    [  ] Overall GUI:
          - In considering the possibility of letting gnuplot be the preferred
            graphical renderer, and the possibility that the data table is
            displayed in a different window than the graphs for the case that
            gnuplot is not chosen/available, se if the actual single window
            display can be provided a 'drag handle' for the main vertical
            division, for the user to jog and adjust the relative size between
            the data table and graph.
          - Eliminate menus, or provide them with appropriate functionality
            (check what they actually do, first).
          - Provide an option to check the console output (useful when the
            experiment is lengthy, and convergence messages appear; the
            combination of this with a Stop button would be practical in my
            opinion).


=== John McCutchan ============================================================

[  ] Rename dataset -> datatable

[  ] Regression testing

[ok] GUI
(G.S. 2009) [ok] Hook up dataset loading/saving to GUI
```

```
[  ] Yield Functions
        [ ] More of them!

[  ] Experiments
        [ ] More of them!

=== EOF ========================================================================
```

[9.3] MatGen `ChangeLog`

```
================================================================================
2009/02/06  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * Doxyfile         MatGen 1.1, 1.0 and 0.2 Doxyfiles updated to ignore .svn/
                    folders, and to avoid generating man/ documentation.

 * build_matgen_mills.sh    Build scripts updated, and changes propagated to
 * build_matgen_macbook.sh  prior versions 1.0 and 0.2.
                            NOTE: Scripts mills_build.sh and macbook_build.sh
                                  were replaced by these.

 * doc_matcalc.sh   Doxygen documentation generation script created, that
                    produces a .tar file under the doc/ folder, containing the
                    html files; the .tar file unpacks to doc/html folder,
                    but this is now ignored by SVN, so the user must unpack the
                    files 'by hand'. Script was propagated to prior versions
                    1.0 and 0.2.
                    NOTE: all html documentation was deleted, and .tar files
                          placed in lieu of it; SVN was mishandling the big
                          number of generated files, with very long filenames.

================================================================================
2009-07-08  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * configure.ac:    Changed 'hardcoded' search for Maple dependencies to warning
                    messages instructing the user to set the following
                    environment variables at the 'configure' script command
                    line:

        AC_MSG_WARN([*** Please specify Maple environment on the command line:])
        AC_MSG_WARN([*** MAPLE_ROOT="/path/to/maple/folders"])
        AC_MSG_WARN([*** OPENMAPLE_CFLAGS="-I\$MAPLE_ROOT/extern/include"])
        AC_MSG_WARN([*** OPENMAPLE_LDFLAGS="-L\$MAPLE_ROOT/bin.*PLATFORM*"])
        AC_MSG_WARN([*** OPENMAPLE_LIBS="-lmaplec -lmaple -lgmp"])

                    Note that OPENMAPLE_CFLAGS and OPENMAPLE_LIBS are set in
                    'configure.ac'; wanings are just for the record.

================================================================================
2009-06-22  Gonzalo Sanchez  <sancheg@mcmaster.ca>

 * configure.ac:    Added all macros suggested by 'autoscan' for all supported
                    platforms:

                    AC_HEADER_STDBOOL
                    AC_HEADER_DIRENT
                    AC_CHECK_FUNCS([strdup])
                    AC_FUNC_CLOSEDIR_VOID
                    AC_TYPE_SIZE_T
                    AC_TYPE_SIGNAL
                    AC_C_CONST

================================================================================
```

[9.4] MatGen TODO

```
=== Gonzalo Sanchez ============================================================

[  ] Check for memory leaks, specially prior to all 'return()' statements,
     where possibly char* were forgot to be freed.
     Check for 'strdup()' functions to detect the char* assignments (and
     eliminate the use of strdup(), due to uportability claims; change for
     manual dynamic memory allocation scheme, as done in MatCalc 'configset'


[  ] GUI menus: fix version and author info in 'About'.


[  ] Change folder name from mapleclient and executable name from gui to matgen;
     provide a CLI interface as matcalc has now.


[  ] Modify the path dependency of the Glade XML interface definition file;
     currently is 'hardcoded' as "../data/gui.glade" which then forces the
     execution of 'gui' exclusively from its installation folder.


[- ] Generate a single 'matgen' executable that can spawn the CLI batch version
     (e.g. either when invoked with a '-b' switch, or implicitly if any function
     name(s) are provided as arguments) or the GUI  version if no argument
     and no option is provided on invocation.


[  ] Improve build options: enable excluding the GUI with switch --without-gui.



[  ] Maple issues

        [  ] ******* IMPORTANT WARNING ********
             Generated material class files with Maple13 on Noll MacBook are
             about one order of magnitude bigger than the original material
             class files, for the same material model.
             They compile OK, but can take up to *** 13 hours *** to build.
             What is worse, *** the simulation results are wrong *** when these
             generated materials are used.
             THIS PROBLEM STILL NEEDS TO BE ADDRESSED; SO FAR NO CLUE TO WHAT
             IS THE REASON BEHIND THIS SITUATION.


        [ok] Fix material generation problem, that either apparently leaves an
            unfinished material class file (containing CodeGenerate[] commands)
            which break matcalc's build, or simplifies the expressions until
            oblivion which causes matcalc to abort execution due to singular
            matrices.
            Tip: start Maple on mills and enter in J2 and then F (for VonMises
            material model); then take whatever partial derivatives of F such
            as \partial F/ \partial sigma_xx etc..  and check the output to
            understand what is happening.

        [ ] Fix problem with Maple searching for one of its libraries under the
            folders pertaining to MatGen (maybe related to following issue)

        [ ] Fix problem that arises at least on <mills.cas.mcmaster.ca> that
            requires the following symlinks for Maple to work:

            src/libgmp.so       -> /usr/local/maple11/bin.X86_64_LINUX/libgmp.so
            src/mfsd            -> /usr/local/maple11/bin.X86_64_LINUX/mfsd
```

```
            license/license.dat -> /usr/local/maple11/license/license.dat


        [ ] Eliminate 'configure.ac' checks for Maple, warn the user of this
            unchecked dependency in the README file, and refer to INSTALL2 for
            installation instructions

            OR

            Leave a simple check for Maple in 'configure.ac' (e.g. check if it
            runs; if it does, just warn at the end of the 'configure' output
            to the console/terminal that if the libraries are not in the default
            search location for the linker and runtime linker, the build will
            fail and the executable would not run either, and refer to
            the INSTALL2 instructions; and if it does not run break the cfg. and
            output an error stating that Maple is required, and point to the
            INSTALL2 instructions thereafter)

            OR

            Wrap the 'hardcoded' checks for Maple so that they can be smoothly
            overriden from the 'configure' script command-line with a
            '--maple-cflags=' switch, or perhaps better,
                '--maple-include-dir="/usr/local/maple/include" '
                '--maple-libs-dir="/usr/local/maple/lib" '

            OR

            Best would be to have an automated check for Maple, just as we have
            for Ruby in 'matcalc/configure.ac'



[  ] Run 'ifnames' on src code (see Autoconf manual)



[  ] Linker switches for Darwin (MacOS X)
        [  ] Check which are the corresponding switches or constructs for the
             dynamic linker 'ld.so' in Darwin, that enable 'hardcoding' into
             the compiled binaries the 'runpath' for dependencies (i.e. which
             is the equivalent switch to '-rpath' or '-R' in Linux and Solaris,
             respectively).
             Also check how to pass it to the static linker 'ld' from the
             compiler driver's command line (for GNU 'gcc', it is the switch
             '-Xlinker': see INSTALL2 file for an example)


=== John McCutchan ====================================================================

[  ] Codegen:
      [  ] Finalize yield interface


[  ] Long Term:
      [  ] Ability to remove constants
      [  ] Ability to set default values for constants
      [  ] Description to each constant that gets passed to gui



=== EOF ===============================================================================
```

[9.5] INSTALL2

```
Additional Installation Instructions
************************************

Informing 'configure' of non-standard locations for dependencies
================================================================

If you have dependencies (header files and/or libraries) in non standard
locations, you can inform the 'configure' script of these by the use of
appropriate environment variables and switches, passed on the command line.


EXAMPLE:

  If using GCC suite and GNU 'ld' linker, the following command will cause the
  'configure' script to generate a Makefile that instructs the compiler to:

  - search for header files in the "/non-std-path/include" folder (in addition
    the default "usr/include" path)

  - in turn, inform the linker 'ld' that additionally to the default search
    paths it should search the "/non-std-path/lib" folder at build time, for
    the libraries "lib_x.so", "lib_y.so" and "lib_z.so"

  - also inform the linker that it should 'hardcode' into the resulting binaries
    the "/non-std-path/lib" search path, in order to enable the runtime linker
    'ld.so' to find the libraries to bind at run-time:


    ./configure CPPFLAGS="-I/non-std-path/include " \
                LDFLAGS="-L/non-std-path/lib \
                         -Xlinker -rpath -Xlinker /non-std-path/lib " \
                   LIBS="-llib_x -llib_y -llib_z "


This will allow the configure script to adapt to any particularities of the
system on which the build is taking place. It is useful in cases such as when:

    * the system administrator has installed some libraries and/or header files
      in a non-standard location, where neither the compiler and/or linker will
      search by default

    * you are linking against a new version of a library or alongside a tool
      on which the software depends, which you do not want to install
      system-wide (e.g. if you are just testing the software to decide if it
      is worth installing it system-wide along with its dependencies)


Alternatively, environment variables such as LD_LIBRARY_PATH or LD_RUN_PATH can
be used to set the libraries search paths, but this is not recommended except
for temporary usage, since it would affect the runtime linking for ALL
applications (e.g. if these variables are set e.g. in the '.profile' file).

You should consult the manuals of your compiler and linkers (e.g.: 'man gcc',
'man ld' and 'man ld.so') to check the specific syntax for the corresponding
switches.
```

```
    NOTE: Even though some linker manuals list that the '-R' option, when
          followed by a directory (as opposed to a file) is interpreted by the
          linker as a runpath to hardcode into the binary for the runtime
          linker, the use of this switch works well only with Sun Solaris
          or OpenSolaris linkers. For GNU linkers, use the '-Xlinker' and
          '-rpath' combination if '-R' does not work. Also note that
          LD_RUN_PATH might not work either, whereas LD_LIBRARY_PATH always
          works as intended.

          With Sun compilers, the runtime path can effectively be informed by
          the '-R' switch; thus, the command line for aforementioned example
          would be:

          ./configure CPPFLAGS="-I/non-std-path/include " \
                      LDFLAGS="-L/non-std-path/lib \
                               -R/non-std-path/lib " \
                         LIBS="-llib_x -llib_y -llib_z "



Switches
========

On GCC and Sun compilers, the '-I' switch is used to inform the compiler of a
non standard location in where to search for header files at compile/link time.

The '-L' switch is used to inform the linker of a non standard location in where
to search for libraries at  link time, and the '-l' switch to inform the linker
of the corresponding library names.

If using GCC suite and the GNU 'ld' linker, the '-Xlinker' switch is used to
inform the compiler driver that it has to pass the the following option to the
linker. In the case of options which require an argument, the switch '-Xlinker'
has to be given twice: the first instance informs the compiler driver of the
option itself that must be passed on to the linker (e.g. '-rpath') and the
second instance the actual argument for that option  (e.g. the runpath).

With the GNU linker the switch '-rpath' will make the linker 'hardcode' into
the resulting binary or binaries any non standard location (paths) in which the
runtime linker 'ld.so' can search for libraries to bind at runtime.

With the Sun linker, the runpath is 'hardcoded' with the '-R' switch.
```

```
Environment variables
=====================

From the Autoconf Manual, for version 2.63 (9 September 2008), the usage of the
pertinent environment variables is as follows:

  CFLAGS: Debugging and optimization options for the C compiler.
  -------------------------------------------------------------
    If a compiler option affects only the behaviour of the preprocessor (e.g.,
    '-D name'), it should be put into CPPFLAGS instead.

    If it affects only the linker (e.g., '-L directory'), it should be put into
    LDFLAGS instead.

    If it affects only the compiler proper, CFLAGS is the natural home for it.

    If an option affects multiple phases of the compiler, though, matters get
    tricky. One approach to put such options directly into CC, e.g.,
    CC="gcc -m64". Another is to put them into both CPPFLAGS and LDFLAGS, but
    not into CFLAGS.

  CXXFLAGS: Debugging and optimization options for the C++ compiler.
  -----------------------------------------------------------------
    It acts like CFLAGS, but for C++ instead of C.

  CPPFLAGS: Preprocessor options for the C and C++ preprocessors and compilers.
  ---------------------------------------------------------------------------

    This variable's contents should contain options like '-I', '-D', and '-U'
    that affect only the behaviour of the preprocessor. Please see the
    explanation of CFLAGS for what you can do if an option affects other phases
    of the compiler as well.

    Currently, 'configure' always links as part of a single invocation of the
    compiler that also preprocesses and compiles, so it uses this variable also
    when linking programs. However, it is unwise to depend on this behaviour
    because the GNU coding standards do not require it and many packages do not
    use CPPFLAGS when linking programs.

  LDFLAGS: Options for the linker.
  --------------------------------
    This variable's contents should contain options like '-s' and '-L' that
    affect only the behaviour of the linker. Please see the explanation of CFLAGS
    for what you can do if an option also affects other phases of the compiler.

    Don't use this variable to pass library names ('-l') to the linker; use LIBS
    instead.

  LIBS: '-l' options to pass to the linker.
  -----------------------------------------
    The default value is empty, but some Autoconf macros may prepend extra
    libraries to this variable if those libraries are found and provide
    necessary functions.


  FCFLAGS: Debugging and optimization options for the Fortran compiler.
  ---------------------------------------------------------------------


  FFLAGS: Debugging and optimization options for the Fortran 77 compiler.
  -----------------------------------------------------------------------
```

[9.6] matcalc_driver.txt

```
================================================================================
Writing a MatCalc Based Driver Program
================================================================================

MatCalc was designed as a library to be used through a driver program. For a
program using the MatCalc library to function properly, it needs:

    - a  dataset
    - a  test specimen
    - a  test specimen geometry
    - an integrator
    - a  material class
    - an experiment.

Below is a minimal driver program which illustrates how to incorporate it into a
larger program:

 +-------------------------------------------------------------------------------+
 | 1  #include "brick_element.h"                                                 |
 | 2  #include "dataset.h"                                                       |
 | 3  #include "experiment_list.h"                                               |
 | 4  #include "material_class_list.h"                                           |
 | 5  #include "returnmap_integrator.h"                                          |
 | 6                                                                             |
 | 7  static  material_class_list*  mat_list = NULL;                             |
 | 8  static  experiment_list*      exp_list = NULL;                             |
 | 9                                                                             |
 |10  int main (int argc, char **argv)                                           |
 |11  {                                                                          |
 |12      mat_list = new  material_class_list();                                 |
 |13      exp_list = new  experiment_list();                                     |
 |14                                                                             |
 |15      brick_element  brick;                                                  |
 |16      m_vector       xyz = get_brick_element_xyz(<lx>, <ly>, <lz>);          |
 |17                                                                             |
 |18      dataset  ds;                                                           |
 |19      ds.add_constant("ElasticE",  <E>);                                     |
 |20      ds.add_constant("ElasticNu", <Nu>);                                    |
 |21                                                                             |
 |22      experiment_runner*  exp = exp_list->get_instance("Experiment");        |
 |23      material_class*     mat = mat_list->get_instance("Material");          |
 |24                                                                             |
 |25      returnmap_integrator rmi;                                              |
 |26      rmi.integrate(ds, xyz, mat, brick, exp, time, timestep);               |
 |27                                                                             |
 |28      delete exp;      exp = NULL;                                           |
 |29      delete mat;      mat = NULL;                                           |
 |30      delete exp_list; exp_list = NULL;                                      |
 |31      delete mat_list; exp_list = NULL;                                      |
 |32                                                                             |
 |33      return (EXIT_SUCCESS);                                                 |
 |34  }                                                                          |
 +-------------------------------------------------------------------------------+
```

```
Lines 1 through 5 in the example include the necessary header files: brick
element, return map integration algorithm, data table, list of experiments, and
list of material behaviour modules, respectively.

Lines 7 and 8 declare the lists of material classes and experiments, which are
initialized on lines 12 and 13.

Lines 15 and 16 setup the test specimen and the test specimen geometry.

Line 18 initializes the dataset and lines 19 and 20 assign values for the
constants "ElasticE" and "ElasticNu". It is important to note that constants are
stored in the dataset prior to running the experiment.

An experiment runner (a class which handles communication between ruby and C++)
is initialized in line 22 by asking the experiment list for the experiment
named "Experiment".

Similarly, the material model class is initialized in line 23 by asking the
material model list for the material named "Material".

The numerical integration algorithm is initialized in line 25 and it is run in
line 26.

Lines 28 to 31 delete the appropriate objects instances to avoid memory leaks
(and resets the pointers as a safety measure).

---

As the sample driver program illustrates, it is straightforward to get an
instance for most of these objects. Only the material class and experiment
require more work than simply defining some variables. The recommended approach
to obtain a material and experiment instance is to use their respective lists.
There are separate material class and experiment lists and you can get an
instance of any registered material class or experiment simply by passing the
text name to the get_instance() function (lines 22 and 23).

================================================================================
```

[9.7] add_experiment.txt

```
================================================================================
Adding experiments to MatCalc
================================================================================

Experiments in MatCalc are implemented by Ruby scripts, so adding an experiment
consists of developing a Ruby class.

The experiment class provides a way for the numerical integration algorithm to
query which degrees of freedom are fixed and which are free, the prescribed
displacements assigned to each node and the loads at each node.

The numerical integration algorithm also calls the 'setup' function in the
experiment class immediately before beginning an experiment.

At each discrete time point during the run of the simulation, the function
'tick' is called.

Experiment classes must implement the following interface:

- initialize()
    The initialize function is the constructor that all Ruby classes must
    implement.

- setup(dataset)
    The setup method is called once right before the experiment is to begin.
    This method takes a single argument pointing to the dataset that will be
    used in the experiment about to be run (see: Section 3.2.3).
    It has no return value.

- tick(datset, timestep, current_time)
    The tick method is called once at the beginning of each discrete time step
    during the simulation.
    The tick is used to facilitate experiments dynamically modifying their
    state based on the amount of time that has passed since the beginning of
    the experiment.
    This method takes three arguments: An instance of the experiment dataset,
    the change in time and the current elapsed time of the experiment.
    It has no return value.

- get_constraints(dataset, timestep)
    The get constraints method is called whenever the numerical integration
    algorithm needs to check which nodal degree of freedom is free or
    constrained.
    A constrained nodal degree of freedom implies that the displacement value
    obtained from the get_displacements() method must be used in the right hand
    side of Equation 2.23.
    A 24D vector containing boolean values must be returned.
    True is interpreted as constrained.

- get_displacements(dataset, timestep)
    The get_displacements() method is called whenever the numerical integration
    algorithm needs the nodal displacements of the constrained degrees of
    freedom.
    It takes two arguments: An instance of the experiment dataset, and the
    change in time.
    A 24D vector must be returned with the displacement constraints that are
    applied to each nodal degree of freedom. In the case of a purely force
    controlled experiment the zero vector can be returned because the
    prescribed displacements are not used. Entries in this vector are used as
    the right hand side of Equation 2.23.
```

```
- get_forces (dataset)
    The get_forces() method is called whenever the numerical integration
    algorithm needs the nodal force loads.
    It takes a single argument consisting of an instance of the experiment
    dataset.
    A 24D vector must be returned with the force load that is applied to each
    nodal degree of freedom. In the case of a purely displacement controlled
    experiment the zero vector must be returned so that the force vector used
    by the numerical integration algorithm is zero.

- get_constants()
    The get_constants() method is called whenever there is a need for the list
    of constant names that this experiment requires.
    The return value is an array containing strings naming each required
    constant.
    The values assigned to the constants are stored as a mapping between the
    names returned from this function and the values assigned to them in the
    data table. The names are used for the GUI and for consistency checking.

- update_geometry(dataset)
    The update geometry method returns true or false controlling whether or not
    the numerical integration algorithm should update the test specimen
    geometry at the end of each time step.
    When the numerical integrator updates the test specimen geometry it is
    computing the true stress and strain values because it is taking into
    consideration the current configuration of the test specimen and not the
    original.

These methods define an interface that allows a wide variety of experiments to
be implemented, including displacement controlled, load controlled, or a
combination of the two.

Experiments can change their state over the course of time and control whether
or not the experiment subject geometry is updated.

Using the Ruby programming language allows experiments to be as complex as the
user desires.

================================================================================
```

[9.8] add_material.txt

```
================================================================================
Adding material models to MatCalc
================================================================================

All material behaviour classes must be compiled and linked into MatCalc so the
process of adding a new material class is slightly more complex than that of
adding an experiment.

Adding a new material class requires working with the build system used by
MatCalc. MatCalc uses Automake and Autoconf. Commonly referred together as
Autotools or the GNU Build Tools (FSF, 2007).

What follows is a very limited set of instructions to add a new material to the
build system:

    Assuming that the new material name is 'rare', and that its model is defined
    by the class provided in the pair of files 'rare_material.h' and
    'rare_material.cc' (*):

    1. Add your material class files to the build system:

       - Add 'rare_material.h' and 'rare_material.cc' to the MATERIAL_CLASSES
         variable in the 'src/Makefile.am' file.

    2. Add your material class to the internal list:

       - In 'src/material_class_list.cc' include 'rare_material.h'
       - In 'src/material_class_list.cc' increment NUM_MATERIALS.
       - Add the line '{"name_for_rare", rare_material_generator },' to the
         '_default_descriptors[NUM_MATERIALS]' material array.

    3. Build MatCalc

       - Run the 'autogen.sh' shell script (you need to have the 'Autotool'
         toolset for this)
       - Run the generated 'configure' script (Ruby is required)
       - Run 'make' utility to actually build MatCalc package.



================================================================================
Generating a material model with MatGen                                     (*)
================================================================================

The files '*_material.h' and '*_material.cc' can be automatically generated by
MatGen, which relieves the user from hand coding the material class.

MatGen accepts a high level description of the material in terms of four
functions, and generates the necessary code for the class that describes the
material model, to be linked with MatCalc.

Please refer to MatGen documentation (under 'mapleclient/doc/' folder) to learn
how use the automated code generation utility.
```

```
================================================================================
Writing a material model Class by hand                                      (*)
================================================================================

In some cases the user may want to write a material model class by hand instead
of using MatGen. A material class written by hand can be more efficient and
handle some more complex constitutive equations than MatGen can generate code
for.

The material class interface includes the following functions:

- array of string get_constants()
    This function must return a vector of strings naming all the constants that
    this material class needs to function. The names are used for the GUI and
    consistency checks.

- scalar KappaF(dataset, epsilonVP)
    This function must return the scalar value obtained from evaluating the
    function.

- scalar F(dataset, stress, kappa)
    This function must return the scalar value obtained from evaluating the F
    function.

- scalar Q(dataset, stress)
    This function must return the scalar value obtained from evaluating the Q
    function.

- scalar get_gamma(dataset)
    This function must return the scalar value of gamma.

- scalar PhiF(dataset, F)
    This function must return the scalar value obtained from evaluating the Phi
    function.

- vector depsilonvp(dataset, dt, F, stress, kappa)
    This function must return the vector value obtained from evaluating
    DeltaEpsilon^vp. This vector is defined in Equation 2.15.

- vector depsilonvpNL(dataset, dt, F, stress, kappa, lambda)
    This function must return the vector value obtained from evaluating
    Delta Epsilon^vp. It takes a numerical version of lambda.

- vector dstressvp(dataset, dt, F, stress, kappa, epsilonVP)
    This function must return the vector value obtained from evaluating
    DeltaSigma^vp. This vector is defined in Equation 2.25.

- vector dstressvpNL(dataset, dt, F, stress, kappa, epsilonVP, lambda)
    This function must return the vector value obtained from evaluating
    DeltaSigma^vp. This vector is defined in Equation 2.25. It takes a
    numerical version of lambda.

- matrix get_De(dataset)
    This function must return the 6D elastic matrix defined in Equation 2.13.

- matrix get_Dvp(dataset, dt, F, stress, kappa, epsilonVP)
    This function must return the 6D Viscoplastic matrix defined in Equation
    2.24.

- matrix get_RM_Jacobian(dataset, dt, F, kappa, stress, lambda, epsilonVP)
    This function must return the Jacobian matrix used by the returnmap stress
    correction algorithm. This matrix is defined in Equation 2.36.

- vector get_RM_F(dataset, dt, F, stress, dstrain, dstress, lambda)
    This function must return the F vector used by the returnmap stress
    correction algorithm. This vector is defined in Equation 2.35.
```

```
Types of function arguments to material class:

    Argument Name   Type
    -------------   ---------
    dataset         dataset
    epsilonVP       m_vector
    stress          m_vector
    kappa           scalar
    dt              scalar
    F               scalar
    lambda          scalar
    dstress         m_vector
    dstrain         m_vector


================================================================================
Sample hand-coded material .h file
--------------------------------------------------------------------------------

#ifndef __HAND_YIELD_H
#define __HAND_YIELD_H

#include <string>
#include <vector>
#include <cmath>
#include "material_class.h"
#include "dataset.h"
#include "m_vector.h"
#include "m_matrix.h"


class hand_material : public material_class
{
public:
    hand_material () {};
    virtual ~hand_material () {} ;


    /* ----- Gradients ---------------------------------------------------- */
    virtual m_vector dFdSigma (const dataset& _dset, const m_vector _stress,   \
                                                          scalar _kappa) = 0;
    virtual m_vector dQdSigma (const dataset& _dset, const m_vector _stress)   \
                                                                       = 0;
    virtual scalar   dFdKappa (const dataset& _dset, const m_vector _stress,   \
                                                          scalar _kappa) = 0;
    virtual m_vector dKappaFdEpsilonVP (const dataset& _dset, const m_vector   \
                                                            _epsilonvp) = 0;
    virtual scalar   dPhiFdF (const dataset& _dset, scalar _F) = 0;


    /* ----- Constants ---------------------------------------------------- */
    virtual std::vector<std::string> get_constants () = 0;


    /* ----- Constitutive Equation ---------------------------------------- */
    virtual scalar KappaF (const dataset& _dset, m_vector _epsilonvp) = 0;
    virtual scalar F (const dataset& _dset, m_vector _stress, scalar _kappa)   \
                                                                       = 0;
    virtual scalar Q (const dataset& _dset, m_vector _stress) = 0;
    virtual scalar PhiF (const dataset& _dset, scalar _F) = 0;
    virtual scalar get_gamma (const dataset& _dset) = 0;
```

```
    /* ----- Expressions -------------------------------------------------- */
    scalar C1 (const dataset& _dset, scalar _lambdap, scalar _dt, scalar _he,  \
                                                                  scalar _hp);
    scalar He (const dataset& _dset, m_vector _stress, scalar _kappa);
    scalar Hp (const dataset& _dset, m_vector _stress, scalar _kappa, m_vector \
                                                                   _epsilonvp);
    scalar lambda (const dataset& _dset,  scalar _F);
    scalar lambdap (const dataset& _dset, scalar _F);

    m_vector depsilonvp (const dataset& _dset, scalar _dt, scalar _F, m_vector \
                                                    _stress, scalar _kappa);
    m_vector dstressvp (const dataset& _dset, scalar _dt, scalar _F, m_vector  \
                               _stress, scalar _kappa, m_vector _epsilonvp);

    m_vector depsilonvpNL (const dataset& _dset, scalar _dt, scalar _F,        \
                           m_vector _stress, scalar _kappa, scalar _lambda);
    m_vector dstressvpNL (const dataset& _dset, scalar _dt, scalar _F,         \
      m_vector _stress, scalar _kappa, m_vector _epsilonvp, scalar _lambda);


    /* ----- Constitutive Matrices ---------------------------------------- */
    m_matrix get_De (const dataset& _dset);
    m_matrix get_Dvp (const dataset& _dset, scalar _dt, scalar _F, m_vector    \
                               _stress, scalar _kappa, m_vector _epsilonvp);

};

#endif



================================================================================
Sample hand-coded material .cc file
--------------------------------------------------------------------------------

#include "hand_material.h"

scalar
hand_material::C1 (const dataset& _dset, scalar _lambdap, scalar _dt, scalar   \
                                                          _he, scalar _hp)
{
    scalar _C1 = 1 + _lambdap * _dt * (_he + _hp);
    return 1.0 / _C1;
}

scalar
hand_material::He (const dataset& _dset, m_vector _stress, scalar _kappa)
{
    return dFdSigma(_dset, _stress, _kappa).dot(get_De(_dset) *                \
                                            dQdSigma(_dset, _stress));
}

scalar
hand_material::Hp (const dataset& _dset, m_vector _stress, scalar _kappa,      \
                                                 m_vector _epsilonvp)
{
    return -dFdKappa(_dset, _stress, _kappa) * (dKappaFdEpsilonVP (_dset,      \
                              _epsilonvp).dot(dQdSigma (_dset, _stress)));
}

scalar
hand_material::lambda (const dataset& _dset,  scalar _F)
{
    return get_gamma(_dset) * PhiF(_dset, _F);
}
```

```
scalar
hand_material::lambdap (const dataset& _dset, scalar _F)
{
    return get_gamma(_dset) * dPhiFdF (_dset, _F);
}

m_vector
hand_material::depsilonvp (const dataset& _dset, scalar _dt, scalar _F,        \
                                          m_vector _stress, scalar _kappa)
{
    return lambda (_dset, _F) * _dt * dQdSigma (_dset, _stress);
}

m_vector
hand_material::dstressvp (const dataset& _dset, scalar _dt, scalar _F,         \
                       m_vector _stress, scalar _kappa, m_vector _epsilonvp)
{
    scalar _lambdap = lambdap (_dset, _F);
    scalar _he = He(_dset, _stress, _kappa);
    scalar _hp = Hp(_dset, _stress, _kappa, _epsilonvp);
    return _dt * lambda(_dset, _F) * C1(_dset, _lambdap, _dt, _he, _hp) *      \
                              * (get_De(_dset) * dQdSigma(_dset, _stress));
}

m_vector
hand_material::depsilonvpNL (const dataset& _dset, scalar _dt, scalar _F,      \
                            m_vector _stress, scalar _kappa, scalar _lambda)
{
    return _lambda * _dt * dQdSigma (_dset, _stress);
}

m_vector
hand_material::dstressvpNL (const dataset& _dset, scalar _dt, scalar _F,       \
       m_vector _stress, scalar _kappa, m_vector _epsilonvp, scalar _lambda)
{
    scalar _lambdap = lambdap (_dset, _F);
    scalar _he = He(_dset, _stress, _kappa);
    scalar _hp = Hp(_dset, _stress, _kappa, _epsilonvp);
    return _dt * _lambda * C1(_dset, _lambdap, _dt, _he, _hp) *                \
                              * (get_De(_dset) * dQdSigma(_dset, _stress));
}
```

```
m_matrix
hand_material::get_De (const dataset& _dset)
{
    scalar ElasticE = _dset.get_constant ("ElasticE");
    scalar ElasticNu = _dset.get_constant ("ElasticNu");
    m_matrix De(6,6);

    De[De.index(0,0)] = (1-ElasticNu);
    De[De.index(1,1)] = (1-ElasticNu);
    De[De.index(2,2)] = (1-ElasticNu);
    De[De.index(3,3)] = (1-2*ElasticNu)/2;
    De[De.index(4,4)] = (1-2*ElasticNu)/2;
    De[De.index(5,5)] = (1-2*ElasticNu)/2;

    De[De.index(0,1)] = ElasticNu;
    De[De.index(0,2)] = ElasticNu;
    De[De.index(1,0)] = ElasticNu;
    De[De.index(2,0)] = ElasticNu;
    De[De.index(2,1)] = ElasticNu;
    De[De.index(1,2)] = ElasticNu;

    scalar denom = (1+ElasticNu)*(1-2*ElasticNu);
    De *= (ElasticE/denom);
    return De;
}

m_matrix
hand_material::get_Dvp (const dataset& _dset, scalar _dt, scalar _F,           \
                        m_vector _stress, scalar _kappa, m_vector _epsilonvp)
{
    m_matrix I(6,6);
    I.set_identity();
    scalar _lambdap = lambdap (_dset, _F);
    scalar _he = He(_dset, _stress, _kappa);
    scalar _hp = Hp(_dset, _stress, _kappa, _epsilonvp);
    scalar T = _dt * C1(_dset, _lambdap, _dt, _he, _hp) * _lambdap *           \
        * (dQdSigma(_dset, _stress).dot(dFdSigma (_dset, _stress, _kappa)));
    m_matrix De = get_De (_dset);

    return De * (I - T * De);
}
```

```
================================================================================
```

[9.9] generate_material.txt

```
================================================================================
Generating material models with MatGen
================================================================================

MatGen provides bot both a GUI and a CLI tool for generating material models in
an automated way.

The process of using the MatGen GUI to generate a new material behaviour model
consists of the following steps:

    1. Provide the material class name

    2. Provide expressions for F, Q, Kappa, Phi, and gamma (*)

    3. Add any constants that are present in the above expressions

    4. Click the "Generate" button


The process of using the MatGen CLI tool to generate a new material behaviour
model consists of the following steps:

    1. Create an empty ASCII text file in the 'src/functions/' folder for each
       material that you want to generate, and name each file accordingly.

    2. Complete each file as follows (the TEMPLATE file in 'src/functions/'
       provides a reminder of the following structure):

       In line 1 of the file, type the expression for F     (*)
       In line 2 of the file, type the expression for Q     (*)
       In line 3 of the file, type the expression for Kappa (*)
       In line 4 of the file, type the expression for Phi   (*)
       In line 5 of the file, type the expression for gamma (*)

       If you used any constant in the expressions for F, Q, Kappa, Phi, and/or
       gamma, then type each constant name, one per line, in the following lines
       of the file; e.g:

       In line 6 of the file, type the expression for the 1st constant used (*)
       In line 7 of the file, type the expression for the 2nd constant used (*)
       (etc.)

       A sample file might be:
       ----------------------
       (3 * J2)^(1/2)         (expr. for F; uses a macro 'J2', described below)
       (3 * J2)^(1/2)         (expr. for Q; in this case, Q=F)
       0.0                    (expr. for Kappa; in this case, is 'don't care')
       F                      (expr. for Phi; in this case, is the identity)
       (1.0/(2.0 * eta))      (expr. for gamma; in this case, depends on 'eta')
       eta                    (expr. for the only user's constant used above)
       ----------------------

    4. Run the 'batch' utility.


After the material model has been generated, C++ source and header files will
be written with the file name of the material name followed by "_material.h"
and "_material.cc". For instance, if the material name is ViscoElastic, then
the files "ViscoElastic_material.h" and "ViscoElastic_material.cc" would be
generated.

Copy the generated files to 'matcalc/src/' folder, and follow the instructions
(under 'matcalc/doc/' folder) for adding a new material to MatCalc.
================================================================================
```

```
MatGen DSL                                                                    (*)
-------------------------------------------------------------------------------

    To define the functions F, Q, Kappa, Phi, and the constant gamma as input
    to MatGen, a Domain Specific Language (DSL) must be used.

    In MatGen's DSL, each of the functions F, Q, Kappa, and Phi is defined by a
    single expression <expr>.

    The Backus-Naur form for building <expr> follows (extended with some
    regular expression operations):


      <expr> --> <num> |

      (<expr>) |

      <expr> ^ <expr> |

      <expr> * <expr> |

      <expr> / <expr> |

      <expr> + <expr> |

      <expr> - <expr> |

      - <expr> |

      sin(<expr>) | arcsin(<expr>) | cos(<expr>) | arccos(<expr>) |

      ln(<expr>) | log(<expr>)|

      <simulation-variable> | <simulation-variable-macros> |
        | <user-defined-constants>

      <num> --> [<sign>]<digit>+[<decimal-point><digit>+]

      <sign> --> +|-

      <decimal-point> --> .

      <string> --> <character>+

      <character> --> 'a'...'z'|'A'...'Z'

      <digit> --> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

      <simulation-variable> -->  <simulation-variable-F> |
        | <simulation-variable-Q> | <simulation-variable-Kappa> |
        | <simulation-variable-Phi>

      <simulation-variable-F> --> 'Kappa' | <simulation-variable-stress> |
        | <simulation-variable-stress-macros>

      <simulation-variable-Q> --> <simulation-variable-stress> |
        | <simulation-variable-stress-macros>

      <simulation-variable-Kappa> --> <simulation-variable-vp-strain> |
        | <simulation-variable-vp-strain-macros>

      <simulation-variable-Phi> --> 'F'

      <simulation-variable-stress> --> 'SigmaXX' | 'SigmaYY' | 'SigmaZZ' |
        | 'SigmaXY' | 'SigmaYZ' | 'SigmaXZ'
```

```
  <simulation-variable-stress-macros> -->  'Sxx' | 'Syy' | 'Szz' | 'Sxy' |
    | 'Syz' | 'Sxz' | 'Sm' | 'J2' | 'J3' | 'q'

  <simulation-variable-vp-strain> --> 'EpsilonVPXX' | 'EpsilonVPYY' |
    | 'EpsilonVPZZ' | 'EpsilonVPXY' | 'EpsilonVPYZ' | 'EpsilonVPXZ'

  <simulation-variable-vp-strain-macros> --> 'EVPxx' | 'EVPyy' | 'EVPzz' |
    | 'EVPxy' | 'EVPyz' | 'EVPxz' | 'J2EVP' | 'EqVP'

  <user-defined-constants> --> <string>


Operators follow the following precedence: (), ^, , /, +, -.

The extended BNF syntax includes [...] which denotes an optional portion
of <expr> and + which denotes one or more repetitions.

'Token' signifies a terminal token.

Note that some of the simulation-variable and simulation-variable-macros
defined are only available when used in a function that accepts them as
arguments:

* Variables available to function F:
  ----------------------------------
  Input Name      Input Description
  ----------------------------------
  SigmaXX         Stress xx
  SigmaYY         Stress yy
  SigmaZZ         Stress zz
  SigmaXY         Stress xy
  SigmaYZ         Stress yz
  SigmaXZ         Stress xz
  Kappa           Hardening parameter


* Variables available to function Q:
  ----------------------------------
  Input Name      Input Description
  ----------------------------------
  SigmaXX         Stress xx
  SigmaYY         Stress yy
  SigmaZZ         Stress zz
  SigmaXY         Stress xy
  SigmaYZ         Stress yz
  SigmaXZ         Stress xz


* Variables available to function Kappa:
  ---------------------------------------
  Input Name      Input Description
  ---------------------------------------
  EpsilonVPXX     Strain xx (viscoplastic)
  EpsilonVPYY     Strain yy (viscoplastic)
  EpsilonVPZZ     Strain zz (viscoplastic)
  EpsilonVPXY     Strain xy (viscoplastic)
  EpsilonVPYZ     Strain yz (viscoplastic)
  EpsilonVPXZ     Strain xz (viscoplastic)
```

```
* Variables available to function Phi:
  ------------------------------------
  Input Name      Input Description
  ------------------------------------
  F               Function F


* Macros description:
  ------------------------------------------------------------------
  Macro Name      Expansion
  ------------------------------------------------------------------
  Sxx             (SigmaXX - (1/3)*(SigmaXX + SigmaYY + SigmaZZ))
  Syy             (SigmaYY - (1/3)*(SigmaXX + SigmaYY + SigmaZZ))
  Szz             (SigmaZZ - (1/3)*(SigmaXX + SigmaYY + SigmaZZ))
  Sxy             (SigmaXY)
  Syz             (SigmaYZ)
  Sxz             (SigmaXZ)
  Sm              (1/3) * (SigmaXX + SigmaYY + SigmaZZ)
  J2              (1/2) * (Sxx^2 + Syy^2 + Szz^2 +  \
                     2  * (Sxy^2 + Sxz^2 + Syz^2))
  J3              SigmaXX * SigmaYY * SigmaZZ -     \
                  SigmaXX * SigmaYZ^2 +             \
                  2 * SigmaXY * SigmaYZ * SigmaXZ - \
                  SigmaXY^2 * SigmaZZ -             \
                  SigmaYY * SigmaXZ^2
  q               (3 * J2)^(1/2)
  EVPxx           (EpsilonVPXX - \
                  (1/3)*(EpsilonVPXX + EpsilonVPYY + EpsilonVPZZ))
  EVPyy           (EpsilonVPYY - \
                  (1/3)*(EpsilonVPXX + EpsilonVPYY + EpsilonVPZZ))
  EVPzz           (EpsilonVPZZ - \
                  (1/3)*(EpsilonVPXX + EpsilonVPYY + EpsilonVPZZ))
  EVPxy           (EpsilonVPXY)
  EVPyz           (EpsilonVPYZ)
  EVPxz           (EpsilonVPXZ)
  J2EVP           (1/2) * (EVPxx^2 + EVPyy^2 + EVPzz^2 + \
                      2 * (EVPxy^2 + EVPxz^2 + EVPyz^2))
  EqVP            (((4/3) * J2EVP)^1/2)
  ------------------------------------------------------------------


The above macros are defined in the file 'mapleclient/data/yieldf.maple',
which contents are copied below as reference.
```

```
================================================================================
mapleclient/data/yieldf.maple
--------------------------------------------------------------------------------
# This script assumes that CGF, CGQ, CGKappaF, CGPhiF, CGGamma are defined before
being run.
# The outputs in terms of Maple variables are:
# CGC1
# CGHe
# CGHp
# CGLambda
# CGLambdaP
# CGDeltaEpsilonVP
# CGDeltaSigmaVP
# CGDvp

with(LinearAlgebra);

# Invariants
Sxx := (SigmaXX - (1/3)*(SigmaXX + SigmaYY + SigmaZZ));
Syy := (SigmaYY - (1/3)*(SigmaXX + SigmaYY + SigmaZZ));
Szz := (SigmaZZ - (1/3)*(SigmaXX + SigmaYY + SigmaZZ));
Sxy := (SigmaXY);
Syz := (SigmaYZ);
Sxz := (SigmaXZ);
Sm := -(1/3)*(SigmaXX + SigmaYY + SigmaZZ);
J2 :=
26/9*SigmaXX*SigmaYY*SigmaZZ+7/9*SigmaYY*SigmaXX^2+7/9*SigmaXX*SigmaYY^2+7/9*SigmaXX^
2*SigmaZZ+4/27*SigmaXX^3+7/9*SigmaXX*SigmaZZ^2-
4/3*SigmaXX*SigmaYZ^2+7/9*SigmaYY^2*SigmaZZ+4/27*SigmaYY^3+7/9*SigmaYY*SigmaZZ^2-
1/3*SigmaYY*SigmaYZ^2+4/27*SigmaZZ^3-1/3*SigmaZZ*SigmaYZ^2+2*SigmaXY*SigmaYZ*SigmaXZ-
4/3*SigmaXY^2*SigmaZZ-1/3*SigmaXY^2*SigmaXX-1/3*SigmaXY^2*SigmaYY-
4/3*SigmaYY*SigmaXZ^2-1/3*SigmaXZ^2*SigmaXX-1/3*SigmaXZ^2*SigmaZZ;
J3 := SigmaXX*SigmaYY*SigmaZZ-SigmaXX*SigmaYZ^2+2*SigmaXY*SigmaYZ*SigmaXZ-
SigmaXY^2*SigmaZZ-SigmaYY*SigmaXZ^2;
q := (3 * J2)^(1/2);
EVPxx := (EpsilonVPXX - (1/3)*(EpsilonVPXX + EpsilonVPYY + EpsilonVPZZ));
EVPyy := (EpsilonVPYY - (1/3)*(EpsilonVPXX + EpsilonVPYY + EpsilonVPZZ));
EVPzz := (EpsilonVPZZ - (1/3)*(EpsilonVPXX + EpsilonVPYY + EpsilonVPZZ));
EVPxy := (EpsilonVPXY);
EVPyz := (EpsilonVPYZ);
EVPxz := (EpsilonVPXZ);
J2EVP := (1/2) * (EVPxx^2 + EVPyy^2 + EVPzz^2 + 2 * (EVPxy^2 + EVPxz^2 + EVPyz^2));
EqVP := ((4/3) * J2EVP)^(1/2);

Ded := (1+ElasticNu)*(1-2*ElasticNu);
De := < < (1-ElasticNu)  | ElasticNu      | ElasticNu     | 0                | 0
| 0                >,
        < ElasticNu      | (1-ElasticNu)  | ElasticNu     | 0                | 0
| 0                >,
        < ElasticNu      | ElasticNu      | (1-ElasticNu) | 0                | 0
| 0                >,
        < 0              | 0              | 0             | (1-2*ElasticNu)/2 | 0
| 0                >,
        < 0              | 0              | 0             | 0                | (1-
2*ElasticNu)/2 | 0                  >,
        < 0              | 0              | 0             | 0                | 0
| (1-2*ElasticNu)/2 >
      >;
De := ElasticE/Ded * De;

dFdSigmaT := < diff(CGF, SigmaXX) | diff(CGF, SigmaYY) | diff(CGF, SigmaZZ) |
diff(CGF, SigmaXY) | diff(CGF, SigmaYZ) | diff(CGF, SigmaXZ) >;
dFdSigma := Transpose(dFdSigmaT);
dQdSigmaT := < diff(CGQ, SigmaXX) | diff(CGQ, SigmaYY) | diff(CGQ, SigmaZZ) |
diff(CGQ, SigmaXY) | diff(CGQ, SigmaYZ) | diff(CGQ, SigmaXZ) >;
dQdSigma := Transpose(dQdSigmaT);
```

```
dFdKappa := diff(CGF, Kappa);
dKappadEpsilonVPT := < diff(CGKappaF, EpsilonVPXX) | diff(CGKappaF, EpsilonVPYY) |
diff(CGKappaF, EpsilonVPZZ) | diff(CGKappaF, EpsilonVPXY) | diff(CGKappaF,
EpsilonVPYZ) | diff(CGKappaF, EpsilonVPXZ) >;

CGDe := De;
CGHe := VectorMatrixMultiply(dFdSigmaT, De) .  dQdSigma;
CGHe := simplify(CGHe);
CGHp := dFdKappa * (dKappadEpsilonVPT . dQdSigma);
CGHp := simplify(CGHp);
CGLambda := CGGamma * CGPhiF;
CGLambda := simplify(CGLambda);
CGLambdaP := diff(CGLambda, F);
CGLambdaP := simplify(CGLambdaP);
CGC1 := 1.0 / (1 + CGLambdaP * dt * (CGHe + CGHp));
CGC1 := simplify(CGC1);
CGDeltaEpsilonVP := dt * CGLambda * dQdSigmaT;
CGDeltaEpsilonVP := simplify(CGDeltaEpsilonVP);
CGDeltaEpsilonVPNL := dt * Lambda * dQdSigmaT;
CGDeltaEpsilonVPNL := simplify(CGDeltaEpsilonVPNL);
CGDeltaSigmaVP := dt * CGC1 * CGLambda * (De . dQdSigma);
CGDeltaSigmaVP := simplify(CGDeltaSigmaVP);
CGDeltaSigmaVPNL := dt * CGC1 * Lambda * (De . dQdSigma);
CGDeltaSigmaVPNL := simplify(CGDeltaSigmaVPNL);
CGDvpSubTerm := dt * CGC1 * CGLambdaP * (dQdSigmaT . dFdSigma);
#CGDvpSubTerm := simplify(CGDvpSubTerm); -- maple gets stuck here with complex
equations
CGDvp := De - MatrixMatrixMultiply(De, CGDvpSubTerm * De);
#CGDvp := simplify(CGDvp); -- maple gets stuck here with complex equations

# Return Map J and F
dPhidF := diff(CGPhiF, F);
ddQddSigma := <
                < diff(diff(CGQ, SigmaXX),SigmaXX) | diff(diff(CGQ, SigmaXX),SigmaYY)
| diff(diff(CGQ, SigmaXX),SigmaZZ) | diff(diff(CGQ, SigmaXX),SigmaXY) |
diff(diff(CGQ, SigmaXX),SigmaYZ) | diff(diff(CGQ, SigmaXX),SigmaXZ) >,
                < diff(diff(CGQ, SigmaYY),SigmaXX) | diff(diff(CGQ, SigmaYY),SigmaYY)
| diff(diff(CGQ, SigmaYY),SigmaZZ) | diff(diff(CGQ, SigmaYY),SigmaXY) |
diff(diff(CGQ, SigmaYY),SigmaYZ) | diff(diff(CGQ, SigmaYY),SigmaXZ) >,
                < diff(diff(CGQ, SigmaZZ),SigmaXX) | diff(diff(CGQ, SigmaZZ),SigmaYY)
| diff(diff(CGQ, SigmaZZ),SigmaZZ) | diff(diff(CGQ, SigmaZZ),SigmaXY) |
diff(diff(CGQ, SigmaZZ),SigmaYZ) | diff(diff(CGQ, SigmaZZ),SigmaXZ) >,
                < diff(diff(CGQ, SigmaXY),SigmaXX) | diff(diff(CGQ, SigmaXY),SigmaYY)
| diff(diff(CGQ, SigmaXY),SigmaZZ) | diff(diff(CGQ, SigmaXY),SigmaXY) |
diff(diff(CGQ, SigmaXY),SigmaYZ) | diff(diff(CGQ, SigmaXY),SigmaXZ) >,
                < diff(diff(CGQ, SigmaYZ),SigmaXX) | diff(diff(CGQ, SigmaYZ),SigmaYY)
| diff(diff(CGQ, SigmaYZ),SigmaZZ) | diff(diff(CGQ, SigmaYZ),SigmaXY) |
diff(diff(CGQ, SigmaYZ),SigmaYZ) | diff(diff(CGQ, SigmaYZ),SigmaXZ) >,
                < diff(diff(CGQ, SigmaXZ),SigmaXX) | diff(diff(CGQ, SigmaXZ),SigmaYY)
| diff(diff(CGQ, SigmaXZ),SigmaZZ) | diff(diff(CGQ, SigmaXZ),SigmaXY) |
diff(diff(CGQ, SigmaXZ),SigmaYZ) | diff(diff(CGQ, SigmaXZ),SigmaXZ) >
              >;
#CGRMJ
CGRMJ1 := -IdentityMatrix(6,6) - dt * Lambda * MatrixMatrixMultiply(De, ddQddSigma);
CGRMJ2 := -dt * (De . dQdSigma);
CGRMJ3 := dPhidF * dFdSigma;
CGRMJ4 := -dPhidF * CGHp * dt -1.0/CGGamma;
CGRMJ := <
            < CGRMJ1[1,1] | CGRMJ1[1,2] | CGRMJ1[1,3] | CGRMJ1[1,4] | CGRMJ1[1,5] |
CGRMJ1[1,6] | CGRMJ2[1]>,
            < CGRMJ1[2,1] | CGRMJ1[2,2] | CGRMJ1[2,3] | CGRMJ1[2,4] | CGRMJ1[2,5] |
CGRMJ1[2,6] | CGRMJ2[2]>,
            < CGRMJ1[3,1] | CGRMJ1[3,2] | CGRMJ1[3,3] | CGRMJ1[3,4] | CGRMJ1[3,5] |
CGRMJ1[3,6] | CGRMJ2[3]>,
            < CGRMJ1[4,1] | CGRMJ1[4,2] | CGRMJ1[4,3] | CGRMJ1[4,4] | CGRMJ1[4,5] |
CGRMJ1[4,6] | CGRMJ2[4]>,
```

```
            < CGRMJ1[5,1] | CGRMJ1[5,2] | CGRMJ1[5,3] | CGRMJ1[5,4] | CGRMJ1[5,5] |
CGRMJ1[5,6] | CGRMJ2[5]>,
            < CGRMJ1[6,1] | CGRMJ1[6,2] | CGRMJ1[6,3] | CGRMJ1[6,4] | CGRMJ1[6,5] |
CGRMJ1[6,6] | CGRMJ2[6]>,
            < CGRMJ3[1]   | CGRMJ3[2]   | CGRMJ3[3]   | CGRMJ3[4]   | CGRMJ3[5]   |
CGRMJ3[6]   | CGRMJ4>
         >;
#CGRMF
CGRMdEpsilonT := < dEpsilonXX | dEpsilonYY | dEpsilonZZ | dEpsilonXY | dEpsilonYZ |
dEpsilonXZ >;
CGRMdEpsilon := Transpose(CGRMdEpsilonT);
CGRMdSigmaT := < dSigmaXX | dSigmaYY | dSigmaZZ | dSigmaXY | dSigmaYZ | dSigmaXZ >;
CGRMdSigma := Transpose(CGRMdSigmaT);
CGRMF1 := (De . CGRMdEpsilon) - CGRMdSigma - dt * Lambda * (De . dQdSigma);
CGRMF2 := CGPhiF - ((Lambda)/CGGamma);
CGRMF := < CGRMF1[1] | CGRMF1[2] | CGRMF1[3] | CGRMF1[4] | CGRMF1[5] | CGRMF1[6] |
CGRMF2 >;
CGRMInitialLambda := (CGC1 * Lambda + CGLambdaP) * dFdSigmaT . De . CGRMdEpsilon;


================================================================================
```

[9.10] test_matcalc.sh

```
#!/bin/bash

DATE=$(date +%Y-%m-%d_%H-%M-%S)
TESTLOG=test_$DATE.log

export VALGRIND_OPTS=" -v --tool=memcheck
                       --memcheck:leak-check=yes
                       --memcheck:show-reachable=yes"


# ----- Run executables through Valgrind; create logs ------------------------
echo ""
echo " *** ["$(date)"] ***"
echo "     Starting Valgrind tests...."
cd   ./src

echo "============================================================" \
                                            >  ../logs/$TESTLOG
echo "***** $(date)"                        >> ../logs/$TESTLOG

EXEC="lists"
VGRINDLOG=valgrind_$DATE.$EXEC.log
echo "============================================================" \
                                            >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                     >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                         ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="FEM_test -h"
VGRINDLOG=valgrind_$DATE.FEM_test-h.log
echo "============================================================" \
                                            >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                     >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                         ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="FEM_test -i Ninput.dat -o Noutput.dat"
VGRINDLOG=valgrind_$DATE.FEM_test-iNinput-oNoutput.log
echo "============================================================" \
                                            >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                     >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                         ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="FEM_test -i input.dat -o Noutput.dat"
VGRINDLOG=valgrind_$DATE.FEM_test-oNoutput.log
echo "============================================================" \
                                            >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                     >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                         ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="FEM_test -i input.dat -o output.dat"
VGRINDLOG=valgrind_$DATE.FEM_test-iinput-ooutput.log
echo "============================================================" \
                                            >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                     >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                         ../logs/$VGRINDLOG >> ../logs/$TESTLOG
```

```
EXEC="matcalc -h"
VGRINDLOG=valgrind_$DATE.matcalc-h.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                       ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="matcalc -v"
VGRINDLOG=valgrind_$DATE.matcalc-v.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                       ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="matcalc -c"
VGRINDLOG=valgrind_$DATE.matcalc-c.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                       ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="matcalc -i Ninput.dat -o Noutput.dat"
VGRINDLOG=valgrind_$DATE.matcalc-iNinput-oNoutput.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                       ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="matcalc -i input.dat -o Noutput.dat"
VGRINDLOG=valgrind_$DATE.matcalc-oNoutput.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                       ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="matcalc -i input.dat -o output.dat"
VGRINDLOG=valgrind_$DATE.matcalc-iinput-ooutput.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                       ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="configset_test"
VGRINDLOG=valgrind_$DATE.$EXEC.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                       ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="Ruby_test"
VGRINDLOG=valgrind_$DATE.$EXEC.log
```

```
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                        ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="dset_diff"
VGRINDLOG=valgrind_$DATE.$EXEC.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                        ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="dset_diff Ndset1.dat Ndset2.dat"
VGRINDLOG=valgrind_$DATE.dset_diff-Ndset1-Ndset2.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                        ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="dset_diff dset1.dat Ndset2.dat"
VGRINDLOG=valgrind_$DATE.dset_diff-Ndset2.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                        ../logs/$VGRINDLOG >> ../logs/$TESTLOG


EXEC="dset_diff dset1.dat dset2.dat"
VGRINDLOG=valgrind_$DATE.dset_diff-dset1-dset2.log
echo "============================================================" \
                                          >> ../logs/$TESTLOG
echo "***** $VGRINDLOG"                   >> ../logs/$TESTLOG
valgrind --log-file-exactly=../logs/$VGRINDLOG ./$EXEC
cat                        ../logs/$VGRINDLOG >> ../logs/$TESTLOG


# ----- grep the log to extract relevan info ---------------------------------
#GREP_OPTS=" --after-context=10"
grep $GREP_OPTS --regexp=^=*$              \
                --regexp=^\*\*\*\*\*\*\*\*       \
                --regexp=SUMMARY           \
                --regexp=malloc/free       \
                --regexp=searching         \
                --regexp=checked           \
                --regexp=All\              \
                --regexp=definitely\ lost\: \
                --regexp=possibly\ lost\:   \
                --regexp=still\ reachable\: \
                --regexp=suppressed\:      \
                           ../logs/$TESTLOG > ../logs/tmp1.log

#GREP_OPTS=" --invert-match"
#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *Memcheck\ *       \
#                            ../logs/tmp1.log > ../logs/tmp2.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *Copyright\ *      \
#                            ../logs/tmp2.log > ../logs/tmp1.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *Using\ *          \
```

```
#                           ../logs/tmp1.log > ../logs/tmp2.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ $                  \
#                           ../logs/tmp2.log > ../logs/tmp1.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *memcheck          \
#                           ../logs/tmp1.log > ../logs/tmp2.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *supp\ *           \
#                           ../logs/tmp2.log > ../logs/tmp1.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *at\ *             \
#                           ../logs/tmp1.log > ../logs/tmp2.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *by\ *           \
#                           ../logs/tmp2.log > ../logs/tmp1.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *Use\ *            \
#                           ../logs/tmp1.log > ../logs/tmp2.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *Invalid\ *        \
#                           ../logs/tmp2.log > ../logs/tmp1.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *[0-9][0-9]*\ *bytes\ * \
#                           ../logs/tmp1.log > ../logs/tmp2.log

#grep $GREP_OPTS --regexp=^..[0-9][0-9]*..\ *[0-9][0-9]*\ *errors\ * \
#                           ../logs/tmp2.log > ../logs/tmp1.log

cat ../logs/tmp1.log > ../logs/$TESTLOG
rm  ../logs/tmp?.log

echo ""
echo " *** ["$(date)"] ***"
echo "     Finished Valgrind tests."
echo "     See $TESTLOG for results."
```

[9.11] test_EvalMapleStatement.cc

```
/* ======================================================================== */
/* Test program for EvalMapleStatement function form OpenMaple C API        */
/* ======================================================================== */


/* ----- MacBook ------------------------------------------------------------ */
/*                                                                           */
/* Environment variables:                                                    */
/*
export MAPLE=/Library/Frameworks/Maple.framework/Versions/13/
DYLD_LIBRARY_PATH=/Library/Frameworks/Maple.framework/Versions/13/bin.APPLE_UNIVERSAL
_OSX/
*/
/*                                                                           */
/* Compile comand:                                                           */
/*
g++ -g -Wall -Wchar-subscripts -Wpointer-arith -Wcast-align -Wsign-compare -Wno-
switch -I./ -I/Library/Frameworks/Maple.framework/Versions/13/extern/include/ -L./
-L/Library/Frameworks/Maple.framework/Versions/13/bin.APPLE_UNIVERSAL_OSX/ -lhf
-lmaple -lmaplec test_EvalMapleStatement.cc libmc.cc -o test_EvalMapleStatement
*/


/* ----- Mills server ------------------------------------------------------- */
/*                                                                           */
/* Environment variables:                                                    */
/*
export MAPLE=/usr/local/maple11/
*/
/*                                                                           */
/* Compile comand:                                                           */
/*
g++ -g -Wall -Wchar-subscripts -Wpointer-arith -Wcast-align -Wsign-compare -Wno-
switch -I./ -I/usr/local/maple11/extern/include -L./
-L/usr/local/maple11/bin.X86_64_LINUX -Xlinker -rpath -Xlinker
/usr/local/maple11/bin.X86_64_LINUX -lmaplec test_EvalMapleStatement.cc libmc.cc -o
test_EvalMapleStatement
*/

/* ------------------------------------------------------------------------ */

#include <cstdio>
#include <cstring>
#include <string>

#include "libmc.h"

/* --- Prototype for local function --- */
static int free_resources();

/* --- Declare file-scope global objects --- */
static char* _error_msg_main   = NULL;
static char* _maple_expression = NULL;
static mapleclient *_mc        = NULL;


int main (int argc, char* argv[]) {

    /* --- Check input and get expression to evaluate --- */
    if (argc < 2) {
      //if (_error_msg_main) {free(_error_msg_main); _error_msg_main = NULL;}
        _error_msg_main = strdup("Usage:
test_EvalMapleStatement \"<expression>\"\n" \
            "where <expression> is a Maple expression terminated with \';\'\n" \
```

```
            "e.g.:\n" \
            "   \"CodeGeneration[C](<expression>,resultname=<var_name>);\"\n" \
          "   \"1.0/3.0;\"\n");
        fprintf(stderr, "%s\n", _error_msg_main);
        free_resources();
        return (EXIT_FAILURE);
    } else {
        _maple_expression = strdup(argv[1]);
    }


    /* --- Instantiate mapleclient --- */
    fprintf(stdout,"*** Creating new mapleclient instance...\n");
    _mc = new mapleclient ();
    if (!_mc) {
      //if (_error_msg_main) {free(_error_msg_main); _error_msg_main = NULL;}
        _error_msg_main = strdup("  * ERR: could not create new mapleclient
instance.");
        fprintf(stderr, "%s\n", _error_msg_main);
        free_resources();
        return (EXIT_FAILURE);
    }


    /* --- Set mapleclient output to stdout --- */
    fprintf(stdout,"*** Setting mapleclient output to stdout...\n");
    _mc->set_outfp_stdout ();
    if (_mc->_error) {
      //if (_error_msg_main) {free(_error_msg_main); _error_msg_main = NULL;}
        _error_msg_main = strdup("  * ERR: could not set mapleclient output.");
        fprintf(stderr, "%s\n", _error_msg_main);
        fprintf(stderr, "%s\n", _mc->_error_msg);
        free_resources();
      return (EXIT_FAILURE);
      }


    /* --- Start maple --- */
    fprintf(stdout,"*** Starting Maple...\n");
    _mc->start_maple (0, NULL);
    if (_mc->_error) {
      //if (_error_msg_main) {free(_error_msg_main); _error_msg_main = NULL;}
        _error_msg_main = strdup("  * ERR: could not start Maple.");
        fprintf(stderr, "%s\n", _error_msg_main);
        fprintf(stderr, "%s\n", _mc->_error_msg);
        free_resources();
        return (EXIT_FAILURE);
    }


    /* --- Reset maple --- */
    fprintf(stdout,"*** Resetting Maple...\n");
    _mc->reset_maple ();
    if (_mc->_error) {
      //if (_error_msg_main) {free(_error_msg_main); _error_msg_main = NULL;}
        _error_msg_main = strdup("  * ERR: could not reset Maple.");
        fprintf(stderr, "%s\n", _error_msg_main);
        fprintf(stderr, "%s\n", _mc->_error_msg);
        free_resources();
        return (EXIT_FAILURE);
    }


    /* --- Eval maple statement --- */
    fprintf(stdout,"*** Evaluating expression: %s\n", _maple_expression);
    _mc->eval (_maple_expression);
```

```
    if (_mc->_error) {
      //if (_error_msg_main) {free(_error_msg_main); _error_msg_main = NULL;}
        _error_msg_main = strdup("  * ERR: could not evaluate expression.");
        fprintf(stderr, "%s\n", _error_msg_main);
        fprintf(stderr, "%s\n", _mc->_error_msg);
        free_resources();
        return (EXIT_FAILURE);
    }


    /* --- Stop maple --- */
    fprintf(stdout,"*** Stopping Maple...\n");
    _mc->stop_maple ();
    if (_mc->_error) {
      //if (_error_msg_main) {free(_error_msg_main); _error_msg_main = NULL;}
        _error_msg_main = strdup("  * ERR: could not stop Maple.");
        fprintf(stderr, "%s\n", _error_msg_main);
        fprintf(stderr, "%s\n", _mc->_error_msg);
        free_resources();
        return (EXIT_FAILURE);
    }


    /* --- Free memory and exit --- */
    fprintf(stdout,"*** Freeing allocated memory...\n");
    free_resources();
    return (EXIT_SUCCESS);
}


/* --- free_resources() ------------------------------------------------------ */
static int free_resources() {
    if (_error_msg_main)   {free(_error_msg_main);   _error_msg_main   = NULL;}
    if (_maple_expression) {free(_maple_expression); _maple_expression = NULL;}
    if (_mc)               {delete _mc;              _mc               = NULL;}

    return (EXIT_SUCCESS);
}



/*--- strdup() replacement -------------------------------------------------- */
//    strdup() is not guaranteed to be portable
//
//    char* str_p = (char*) malloc(strlen("my_string")+1);
//    strcpy(str_p,"my_string"); // no buffer overrun possible
//
//    or
//
//    char* my_string_p;
//    ...
//    strncpy(my_string_p, other_str,len_other_str); // init my_string_p
//    ...
//    char* str_p = (char*) malloc(strlen(my_string_p)+1);
//    strcpy(str_p,my_string_p); // no buffer overrun possible


/* ============================================================================ */
```

[9.12] MatCalc 1.1 file list (Doxygen)

The following table (copied from the HTML Doxygen documentation for MatCalc) lists the files that comprise MatCalc, alongside the brief comments for each of the (documented) files.

File List Here is a list of all files with brief descriptions:	
brick_element.cc [code]	[003]
brick_element.h [code]	[003] Header file for brick_element.cc
cli.cc [code]	[001] CLI frontend for MatCalc FEM engine
cli.h [code]	[001] Header file for cli.cc
configset.cc [code]	[002] Current simulation parameters data structure module [CODE REQUIRES CONVERTING struct INTO class]
configset.h [code]	[003] Header file for configset.cc
datachunk_ruby.cc [code]	[003]
datachunk_ruby.h [code]	[003] Header file for datachunk_ruby.cc [Contains code]
dataset.cc [code]	[002] Current simulation data class module
dataset.h [code]	[003] Header file for dataset.cc [Contains code]
dataset_ruby.cc [code]	[003]
dataset_ruby.h [code]	[003] Header file for dataset_ruby.cc [Contains code]
debug_utils.cc [code]	[004]
debug_utils.h [code]	[003] Header file for debug_utils.cc
DEmatrix.cc [code]	[003]
DEmatrix.h [code]	[003] Header file for DEmatrix.cc
elastic_integrator.cc [code]	[003] Elastic integrator algorithm (only useful for elastic materials; use Return map algorithm, valid for all materials)
elastic_integrator.h [code]	[003] Header file for elastic_integrator.cc
element.h [code]	[003] Header file for <unknown> [HEADER REQUIRES REVISION]
experiment_list.cc [code]	[004]
experiment_list.h [code]	[003] Header file for experiment_list.cc
experiment_runner.cc [code]	[004]
experiment_runner.h [code]	[003] Header file for experiment_runner.cc
fem_gq3.cc [code]	[003]
fem_gq3.h [code]	[003] Header file for fem_gq3.cc
graph_widget-marshallers.c [code]	
graph_widget-marshallers.h [code]	
graph_widget.cc [code]	[004]

graph_widget.h [code]	[003] Header file for graph_widget.cc
gui.cc [code]	[001] Gtk/Glade GUI frontend for MatCalc FEM engine
gui.h [code]	[003] Header file for gui.cc
input.cc [code]	[002] MatCalc setup and data text file input module
input.h [code]	[003] Header file for input.cc
integrator.cc [code]	[003]
integrator.h [code]	[003] Header file for integrator.cc
m_matrix.cc [code]	[003]
m_matrix.h [code]	[003] Header file for m_matrix.cc
m_scalar.cc [code]	[003]
m_scalar.h [code]	[003] Header file for m_scalar.cc
m_vector.cc [code]	[003]
m_vector.h [code]	[003] Header file for m_vector.cc
m_vector_ruby.cc [code]	[003]
m_vector_ruby.h [code]	[003] Header file for m_vector_ruby.cc [contains code]
matcalc.cc [code]	[000] MatCalc executable: wrapper that spawns MatCalc's CLI or GUI depending on the provided command-line switches (Run: 'matcalc -h')
matcalc_driver.cc [code]	[002]
matcalc_driver.h [code]	[003] Header file for matcalc_driver.cc
material_class.cc [code]	[003]
material_class.h [code]	[003] Header file for material_class.cc [HEADER REQUIRES REVISION]
material_class_list.cc [code]	[003]
material_class_list.h [code]	[003] Header file for material_class_list.cc
output.cc [code]	[002] MatCalc setup and data text file output module
output.h [code]	[003] Header file for output.cc
returnmap_integrator.cc [code]	[003] Return map integrator algorithm
returnmap_integrator.h [code]	[003] Header file for returnmap_integrator.cc
ruby_helper.cc [code]	[003]
ruby_helper.h [code]	[003] Header file for ruby_helper.cc [Contains code]
viscoplastic_integrator.cc [code]	[003] Viscoplastic integrator algorithm (use of Return map algorithm in lieu of this is encouraged)
viscoplastic_integrator.h [code]	[003] Header file for viscoplastic_integrator.cc
test/B_test.cc [code]	[test] CLI executable for testing the relative error in operations involving matrices B and J asociated to Uniaxial material tests [DOC REQUIRES REVISION]
test/configset_test.cc [code]	[test] CLI executable to check configset.h|cc modules operation
test/D_test.cc [code]	[test] CLI executable for testing the relative error between a simulation based on a material model, and a closed form solution computation [DOC REQUIRES REVISION]

test/FEM_test.cc [code]	[test] CLI tool for testing MatCalc's FEM algorithm (Run: 'FEM_test -h') [CODE REQUIRES REVISION]
test/Glade_test.cc [code]	[test] CLI executable for checking Glade availability (it is assumed that will trigger system errors in case Gtk/Glade are unavailable) [DOC REQUIRES REVISION]
test/Graph_test.cc [code]	[test] CLI executable for testing MatCalc's GUI graph display
test/K_test.cc [code]	[test] CLI executable for testing the behaviour of matrix K under constraints/displacements [DOC REQUIRES REVISION]
test/MatCalc_drv.cc [code]	[test] CLI executable that performs three not well documented tests [CODE REQUIRES REVISION]
test/Math_test.cc [code]	[test] CLI executable to test Matrix and Vector matematical operations (19 tests)
test/MC_test.cc [code]	[test] CLI executable to test the material_class class (9 tests) [CODE REQUIRES REFACTORING]
test/Ruby_test.cc [code]	[test] CLI executable that checks Ruby interaction with the experiment_runner class and ruby experiment scripts
tools/dset_diff.cc [code]	[tool] CLI tool for testing the relative error in Stress and Strain between two MatCalc datasets
tools/hooke_jeeves.cc [code]	[tool] CLI frontend for the Hooke & Jeeves driven fitting tool for MatCalc's material models parameters [ALPHA VERSION]
tools/lists.cc [code]	[tool] CLI executable that dumps the Material Models built into MatCalc and the Experiments available for MatCalc

Generated on Sun Mar 7 19:52:18 2010 for MatCalc by *1.5.7.1*

[9.13] MatCalc 0.1 memory leak bug description and proposed fix

> **Note**: the content of this section is, in its entirety, based on the information provided by original developer of Virtlab, John McCutchan, who diagnosed the problem and proposed the corresponding fix.

The memory leak reported in MatCalc 0.1 is related to Ruby. The leak is not generated inside of Ruby, but is a side effect from the wrapper class that is used, which in specific points fails to free some allocated memory, and also fails to call certain object's destructors, thus creating a memory leak thereafter.

[9.13.1] Description

Whenever an instance of a native class `T` (a C++ class `T`) is needed inside of Ruby, a `ruby_object<T>` instance is created. For example, tracing the code of `m_vector_ruby`, the following happens:

- `m_vector_ruby::create` is called which then calls `ruby_object<m_vector>::construct()` which then calls `Data_Make_Struct`.
- `Data_Make_Struct` allocates (`ruby_xmalloc`) enough memory for an instance of the native class and then zeros the memory. So in this case, it allocates the memory for a C++ `m_vector` instance and zeros that. (Inside `m_vector` we have two `int` and a `scalar*`. On a 32-bit machine this memory allocation should be 8 bytes)
- `ruby_object<m_vector>::construct()` returns this new chunk of memory back to `m_vector_ruby::create`.

The method `m_vector_ruby::create` looks like this (`[m_vector_ruby.h:55]`):

```
00054     // create a ruby object from a c++ m_vector
00055     RUBY_METHOD create (const m_vector& vector)
00056     {
00057         VALUE object;
00058         m_vector* instance = construct (object);
00059         *instance = vector;
00060         return object;
00061     }
```

When `[00058]` has finished executing, `instance` points to valid memory that has been filled with zeros. In `[00059]` the assignment operator `=` for the `vector` class executes. *The assignment operator, =, will allocate memory to store the values of the vector and copy the values from the* `vector` *variable*.

At this point two leaks have been produced:

- `Data_Make_Struct` takes `(*free)` (a function pointer to the function `free()`[77]) but in the code a `NULL` pointer is passed: `[ruby_helper.h:00168]` `object=Data_Make_Struct(ruby_class_id(),T,0,0,internal)`

  Hence, **all the memory allocated inside `Data_Make_Struct` is never freed**

- **The destructors on the objects constructed inside the `::create()` functions are never called**

Memory leaked from `Data_Make_Struct` calls shouldn't be too large chunks, because it is always related to small allocations (`datachunk`, `m_vector` and `dataset` don't need much memory themselves). But the memory leaks from the `::create()` methods can be potentially huge: inside `dataset` and `m_vector` lots of memory can be allocated and this memory is only ever freed when the destructor is called.

[77] `VALUE Data_Make_Struct (VALUE class, c-type, void (* mark)(), void (* free)(), c-type*")`

[9.13.2] Fix

The fix is straightforward:

- Each `::create()` function should be changed to call the *constructor* instead of the *assignment* operator. That would look like:

```
void* p = construct (object)
m_vector* instance = new (p) m_vector(size);
```

- A `::destroy()` method should be written that would look something like:

```
m_vector* instance = internal(object);
instance->~m_vector(); // call the destructor explicitly
```

- Calls to `::destroy()` should be paired with the existing calls to `::create()`. All of the `ruby_object<T>::create` calls are inside `experiment_runner` methods.
- `Data_Make_Struct` should be passed the standard `free()` function

[9.13.3] Additional information

[9.13.3.1] C++:

`m_vector`

base size = 8 bytes
constructor allocates an array of scalars to the size passed into the constructor "`_elements`"
destructor frees the array of scalars "`_elements`"

`datachunk`

- base size = `sizeof(scalar) * 26` bytes
- constructor doesn't allocate any memory
- destructor doesn't free any memory

`dataset`

- base size = `sizeof(std::vector)+sizeof(std::map)`
- `add_constant` causes memory usage to grow
- `add_datachunk` causes memory usage to grow
- destructor frees all memory allocated through `add_constant` / `add_datachunk.`

[9.13.3.2] C++ / Ruby:

`ruby_object<m_vector>`

- `construct()` allocates base size of `m_vector` memory and zeros it [**C++**]
- `construct()` also allocates a handle inside Ruby (small, and collected by ruby GC) [**Ruby**]
- `create()` then makes a copy of the `m_vector` passed into `create()` [**C++**]

`ruby_object<datachunk>`

- `construct()` allocates base size of `datachunk` memory and zeros it [**C++**]
- `construct()` also allocates a handle inside Ruby (small, and collected by ruby GC) [**Ruby**]
- `create()` then makes a copy of the `datachunk` passed into `create()` [**C++**]

`ruby_object<dataset>`

- `construct()` allocates base size of `dataset` memory and zeros it [**C++**]
- `construct()` also allocates a handle inside Ruby (small, and collected by ruby GC) [**Ruby**]
- `create()` then makes a copy of the `dataset` passed into `create()` [**C++**]

The problem is that the C++ destructors for the `dataset` and `m_vector` classes are never run when they are wrapped inside a `ruby_object<>` C++ class.

[9.13.3.3] Ruby:

Whenever an instance of `dataset`, `datachunk` or `m_vector` is accessed it causes C++ code to be executed and possibly allocate memory inside of a `dataset` or `m_vector`. So, it is the C++ objects that grow and resize, not the Ruby objects. There is a tiny leak from the call to `Data_Make_Struct`, the real leak is because the C++ destructors are never called.

The following example shows how to pass a `dataset` to Ruby so that the `tick()` method can be run:

```
VALUE dataset_value = dataset_ruby::create (ds); [1]
VALUE dt_value = ruby_scalar::create (dt);
VALUE ct_value = ruby_scalar::create (ct);
VALUE notused = ruby_call_method (_instance, "tick", 3, dataset_value, dt_value, ct_value);
```

First, the code allocates memory for a Ruby `VALUE` type. The `VALUE` type is a generic Ruby handle/pointer type: `VALUE dataset_value = dataset_ruby::create (ds)`. It also allocates memory for a native C++ `dataset` type. Both of those allocations are small. The memory used for the `VALUE` type is collected by the Ruby GC and currently the memory used for the native C++ `dataset` type is leaked - but this is small. Inside `dataset_value` there is a pointer to a C++ `dataset` instance, which will be referred as `inner`.

Second, the code copies the contents of `ds` into `inner` - this is where the real memory leak is.

After `[1]` is executed, we have two copies of the `dataset`. One in `ds` and the other in `inner`. The leak happens because the destructor of the `inner` is never called. Evidence of this can be seen in the Valgrind callstacks that look like:

```
=30531= 4,307,040 bytes in 626 blocks are indirectly lost in loss record 17 of 18
=30531=  at 0x4A06019: operator new(unsigned long) (vg_replace_malloc.c:167)
=30531=  by 0x406BDD: __gnu_cxx::new_allocator<datachunk>::allocate(unsigned long, void const*)
            (new_allocator.h:88)
=30531=  by 0x406C04: std::_Vector_base<datachunk, std::allocator<datachunk>>::_M_allocate(unsigned long)
            (stl_vector.h:127)
=30531=  by 0x40CBBA: datachunk* std::vector<datachunk,std::allocator<datachunk>>::_M_allocate_and_copy
            <__gnu_cxx::__normal_iterator<datachunk const*, std::vector<datachunk,
            std::allocator<datachunk>>>> (unsigned long, __gnu_cxx::__normal_iterator<datachunk const*,
            std::vector<datachunk, std::allocator<datachunk>>>, __gnu_cxx::__normal_iterator<datachunk
            const*, std::vector<datachunk, std::allocator<datachunk>>>) (stl_vector.h:767)
=30531=  by 0x40CC65: std::vector<datachunk,
            std::allocator<datachunk> >::operator=(std::vector<datachunk, std::allocator<datachunk>>
            const&) (vector.tcc:141)
=30531=  by 0x40D2D8: dataset::operator=(dataset const&) (dataset.h:167)
=30531=  by 0x40D319: dataset_ruby::create(dataset const&) (dataset_ruby.h:41)
```

So the leak is caused by a call to `dataset_ruby::create` but the memory is allocated inside the *assignment* operator. The proposed fix of adding a `::destroy()` function that explicitly calls the destructor should fix this leak. So, the code inside the experiment runner would be modified to look like:

```
VALUE dataset_value = dataset_ruby::create (ds); [1]
VALUE dt_value = ruby_scalar::create (dt);
VALUE ct_value = ruby_scalar::create (ct);
VALUE notused = ruby_call_method (_instance, "tick", 3, dataset_value, dt_value, ct_value);
dataset_ruby::destroy(dataset_value);
```

> **Note**: There is potentially another class of leaks related to `ruby_object`. The issue is simple, but it requires a good understanding of a C++ objects life cycle, constructors, destructors, copy constructors and assignment operators, and an understanding of the steps involved: allocating memory for an object instance, running the object's constructor, (using the object), running the object's destructor, freeing the allocated memory for the object's instance. C++ `placement new` operator should also be understood.

[9.14] Compilation, Assembly and Linking Overview

Note: The complete contents of this section are adapted from:

http://www.tenouk.com/ModuleW.html.

COMPILERS**, **ASSEMBLERS** and **LINKERS

***Normally the C's program building process involves four stages** and utilizes different 'tools' such as a preprocessor, compiler, assembler, and linker[78].*

At the end there should be a single executable file. Below are the stages that happen in order, regardless of the operating system/compiler; these are graphically illustrated in [Fig. 40].

1. ***Preprocessing** is the first pass of any C compilation. It processes include-files, conditional compilation instructions and macros.*
2. ***Compilation** is the second pass. It takes the output of the preprocessor, and the source code, and generates assembler source code.*
3. ***Assembly** is the third stage of compilation. It takes the assembly source code and produces an assembly listing with offsets. Assembler output is stored in an object file.*
4. ***Linking** is the final stage of compilation. It takes one or more object files or libraries as input and combines them to produce a single (usually executable) file. In doing so, it resolves references to external symbols, assigns final addresses to procedures / functions and variables, and revises code and data to reflect new addresses (a process called relocation).*

OBJECT FILES** and **EXECUTABLE

After the source code has been assembled, it will produce an Object file (e.g. .o, .obj). When linked, produces an executable file.

RELOCATION RECORDS

Because the various object files will include references to each others code and/or data, these shall need to be resolved during the link time. For example in [Fig. 41], *the object file that has* `main()` *includes calls to functions* `funct()` *and* `printf()`. *After linking all of the object files together, the linker uses the relocation records to find all of the addresses that need to be filled in.*

LINKING

The linker actually enables separate compilation. As shown in [Fig. 42], *an executable can be made up of a number of source files which can be compiled and assembled into their object files respectively, independently.*

Adapted from: http://www.tenouk.com/ModuleW.html.

[78] Bear in mind that if you use the IDE type compilers, these processes quite transparent.

The following Figure shows the steps involved in the process of building the C program starting from the compilation until the loading of the executable image into the memory for program running:

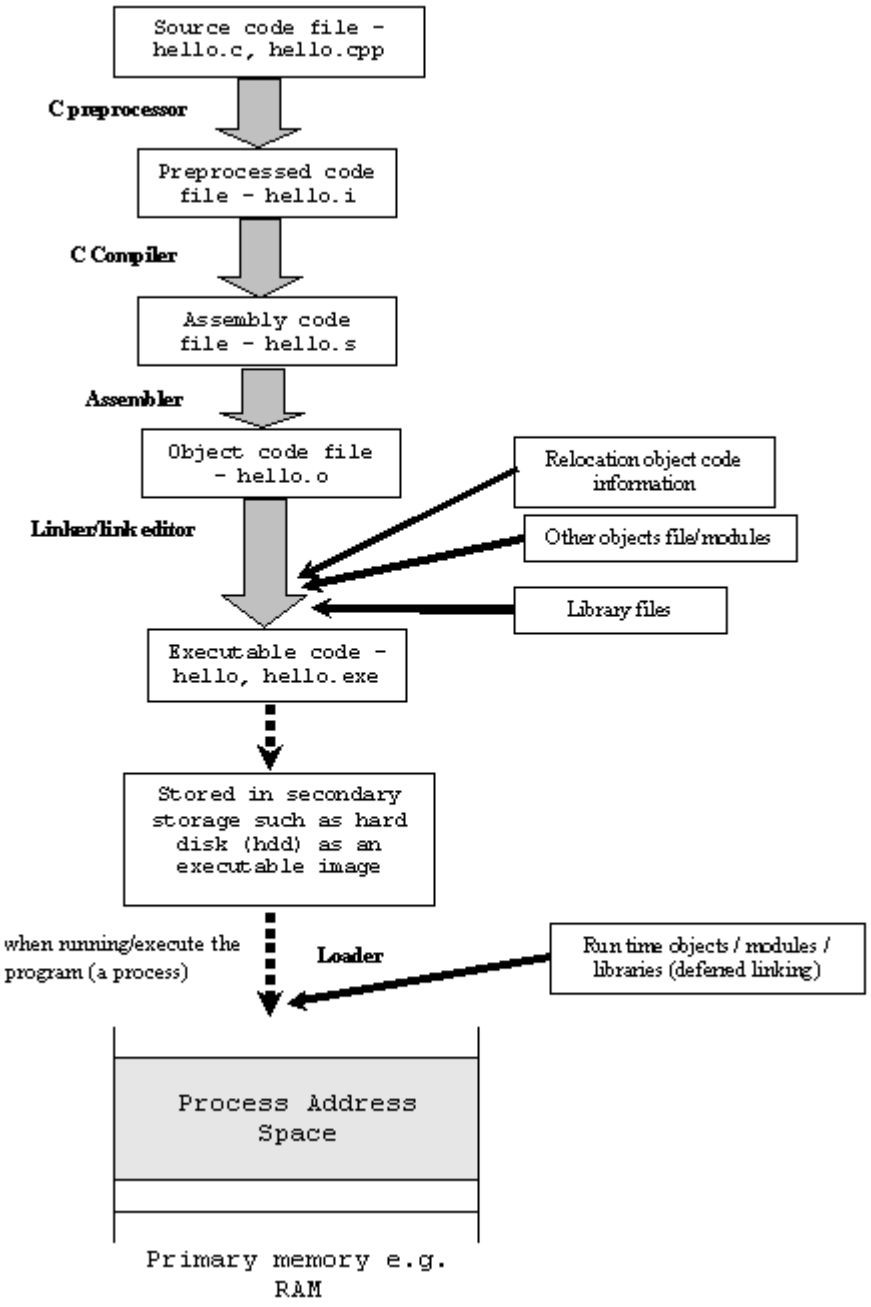

Source code file - hello.c, hello.cpp
C preprocessor
Preprocessed code file - hello.i
C Compiler
Assembly code file - hello.s
Assembler
Object code file - hello.o
Linker/link editor
Relocation object code information
Other objects file/modules
Library files
Executable code - hello, hello.exe
Stored in secondary storage such as hard disk (hdd) as an executable image
when running/execute the program (a process)
Loader
Run time objects / modules / libraries (deferred linking)
Process Address Space
Primary memory e.g. RAM

Fig. 40: Compile, link and execute stages for running program (a process).

[Figure w.1; http://www.tenouk.com/ModuleW.html]

The following figure shows the linker resolving references between object files – including system libraries:

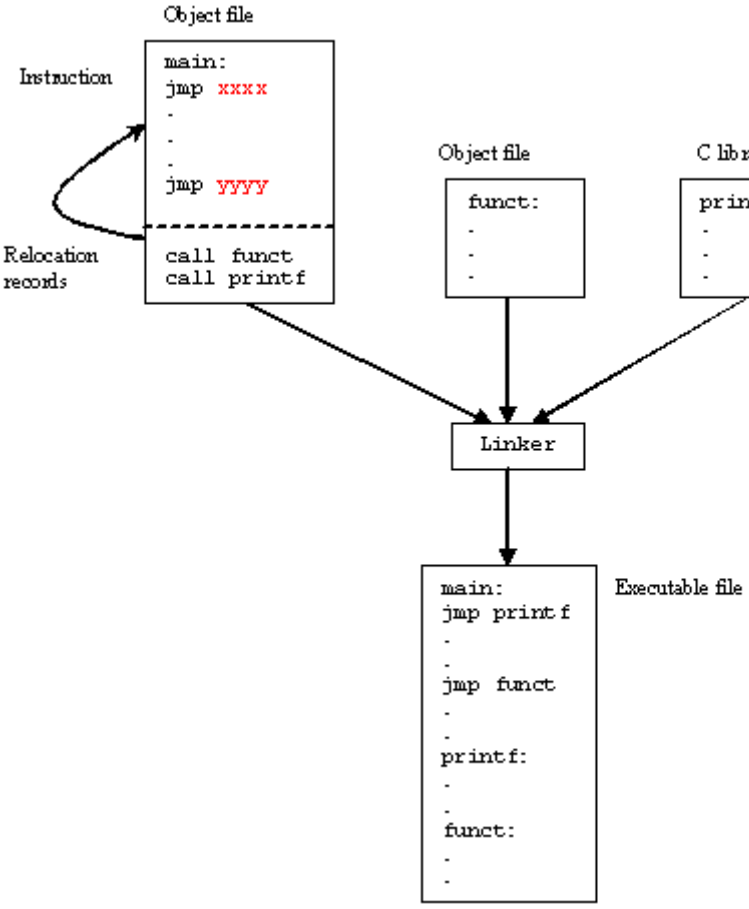


Fig. 41: The relocation record

[Figure w.2: http://www.tenouk.com/ModuleW.html]

The following figure shows how the assembly process can proceed in parallel, followed by a linking process:

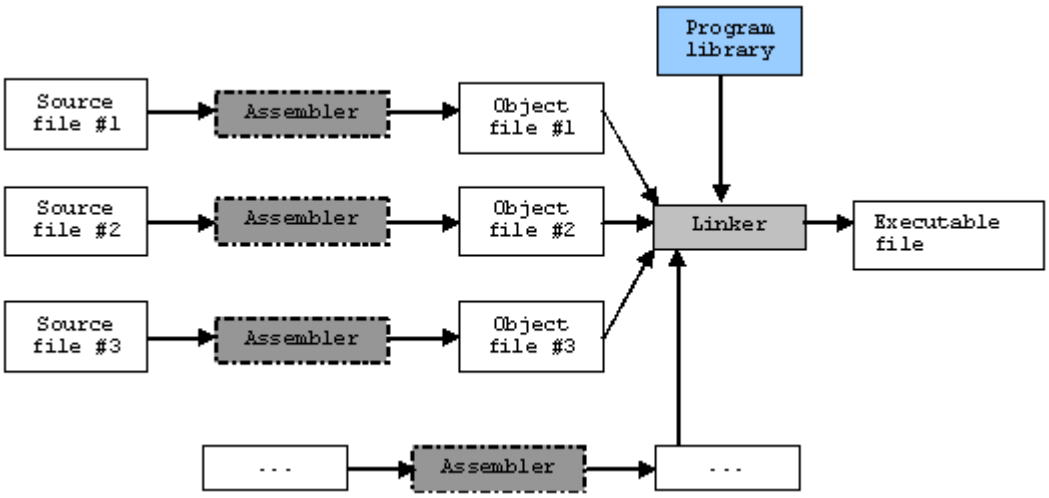


Fig. 42: The object files linking process.

[Figure w.3: http://www.tenouk.com/ModuleW.html]

SHARED OBJECTS

In a typical system, a number of programs will be running. Each program relies on a number of functions, some of which will be standard C library functions, like `printf()`, `malloc()`, `strcpy()`, *etc. and some are non-standard or user defined functions.*

If every program uses the standard C library, it means that each program would normally have a unique copy of this particular library present within it. This would result in wasted resources, and would degrade the efficiency and performance.

Since the C library is common, ***it is better to have each program reference the common, one instance of that library, instead of having each program contain a copy of the library. This is implemented during the linking process, where some of the objects get linked during the link time (static linking) and some others will be linked at run time (deferred/dynamic linking)****.*

STATICALLY LINKED PROGRAM

The term 'statically linked' means that the program and the particular library that it's linked against are combined together by the linker at link time*. This means that the binding between the program and the particular library is fixed and known at link time before the program run. It also means that we can't change this binding, unless we re-link the program with a new version of the library.*

Programs that are linked statically are linked against archives of objects (libraries) that typically have the extension of `.a`. *An example of such a collection of objects is the standard C library,* `libc.a`.

You might consider linking a program statically for example, in cases where you weren't sure whether the correct version of a library will be available at runtime, or if you were testing a new version of a library that you don't yet want to install as shared.

DYNAMICALLY LINKED PROGRAM

The term 'dynamically linked' means that the program and the particular library it references are not combined together by the linker at link time. Instead, the linker places information into the executable that tells the loader which shared object module the code is in and which runtime linker should be used to find and bind the references. *This means that the binding between the program and the shared object is done at runtime that is before the program starts, the appropriate shared objects are found and bound.*

This type of program is called a partially bound executable, because it isn't fully resolved. The linker, at link time, didn't cause all the referenced symbols in the program to be associated with specific code from the library. Instead, the linker simply said something like: "This program calls some functions within a particular shared object, so I'll just make a note of which shared object these functions are in, and go on".

Symbols for the shared objects are only verified for their validity to ensure that they do exist somewhere. The linker stores in the executable program the locations of the external libraries where it found the missing symbols. This effectively defers the binding until runtime.

Programs that are linked dynamically are linked against shared objects that have the extension `.so`. *An example of such an object is the shared object version of the standard C library,* `libc.so`.

The advantage of deferring the linkage of some of the objects/modules during the static linking step until they are finally needed (during the run time) includes:

1. ***Program files (on disk) become much smaller because they need not hold all necessary text and data information**. It is very useful for portability.*
2. ***Standard libraries may be upgraded or patched without every one program need to be re-linked**. This clearly requires some agreed module-naming convention that enables the dynamic linker to find the newest, installed module such as some version specification. Furthermore the distribution of the libraries is in binary form (no source), including dynamically linked libraries (DLLs)[79] and when you change your program you only have to recompile the file that was changed.*

[79] The `.dll` extension and DLL acronym refer to the Microsoft Windows naming convention of shared objects.

3. ***In combination with virtual memory, dynamic linking permits two or more processes to share read-only executable modules such as standard C libraries***. *Using this technique, only one copy of a module needs be resident in memory at any time, and multiple processes, each can executes this shared code (read only). This results in a considerable memory saving, although demands an efficient swapping policy.*

RUNTIME LINKER** AND **SHARED LIBRARY LOADING

***The runtime linker is invoked when a program that was linked against a shared object is started or when a program requests that a shared object be dynamically loaded**.*

***So, the resolution of the symbols can be done at one of the following times**:*

*1. **Load-time dynamic linking** – the application program is read from the disk (disk file) into memory and unresolved references are located. The load time loader finds all necessary external symbols and alters all references to each symbol (all previously zeroed) to memory references relative to the beginning of the program.*

*2. **Run-time dynamic linking** – the application program is read from disk (disk file) into memory and unresolved references are left as invalid (typically zero). The first access of an invalid, unresolved, reference results in a software trap. The run-time dynamic linker determines why this trap occurred and seeks the necessary external symbol. Only this symbol is loaded into memory and linked into the calling program.*

Adapted from: http://www.tenouk.com/ModuleW.html.